UTILIZING THE DIGITAL FINGERPRINT METHODOLOGY FOR SECURE KEY GENERATION

THESIS

Jennifer C. Anilao
2nd Lieutenant, USAF

AFIT/GE/ENG/10-02

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the United States Government.

AFIT/GE/ENG/10-02

UTILIZING THE DIGITAL FINGERPRINT METHODOLOGY FOR SECURE KEY GENERATION

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science in Electrical Engineering

Jennifer C. Anilao
2nd Lieutenant, USAF

March 2010

AFIT/GE/ENG/10-02

UTILIZING THE DIGITAL FINGERPRINT METHODOLOGY FOR SECURE KEY GENERATION

Jennifer C. Anilao, B.S. Electrical Engineering
2nd Lieutenant, USAF

Approved:

//signed// 18 Mar 2010

Dr. Yong Kim (Chairman)

//signed// 18 Mar 2010

Maj LaVern A. Starman, PhD (Member)

//signed// 18 Mar 2010

Maj Eric D. Trias, PhD (Member)

AFIT/GE/ENG/10-02

Abstract

There are several applications in which it would be useful to have the capability to generate a particular number that would be specific to a certain piece of hardware. One application is the key for an encryption algorithm. If a program could be written to produce a unique value for a device, based on its intrinsic properties, this value could be used as an uncloneable key for an encryption algorithm. Currently, there are ways to use the delay characteristics of transistors to create a unique identity for a device. This research examines a way to modify one of these existing methods to instead generate an uncloneable key specific to a particular device.

A key is made up of a series of binary digits. In order to generate a specific key, there must be a way to control whether each bit of the key will be a '0' or a '1'. Theoretically, it should be possible to add buffers into a circuit path to increase the propagation delay through a simple combinational circuit composed of logic gates, which could produce either a glitch of '0' or a '1' on its output. Each bit of the key is an output of one of these simple glitch circuits. The addition of buffers must be selectable to be able to control the amount of buffers added into the path for each bit.

The results detail tests of two configurations for adding a selectable amount of buffers into each glitch circuit in order to induce additional delay. One configuration, the linear selection design, adds up to seven buffers that is equivalent to the binary digits used on the three SELECT lines of a multiplexer. The linear selection implementation produces 30.94% unusable output lines, where a unusable line is defined as one that does not have at least one ‘1’ and one ‘0’ glitch count in response to every buffer count.

The second implementation of buffer selection, referred to as the cascaded design, has eight different quantities of selectable buffers, but they all connect to one multiplexer. Each successive line connects to the previous line and adds a certain number of buffers. The cascaded design generates an average of 53.75% unusable output lines.

Tests were also performed to determine the optimal number of buffers added to each output using the linear buffer selection configuration. Using three input bits to the buffer unit produced 30.94% unusable outputs. Four bits generated nearly 25% more usable outputs, while the use of six bits gave less than a 5% improvement over four bits.

After determining the optimal configuration for controlling the output lines, the user profiles the circuit to deduce the value of the output lines for each number of selectable buffers added to the circuit. The user can then use this information to generate a specific glitch count on each output line, which is passed to an Advanced Encryption Standard algorithm as its key. This research uses a 128-bit key; however, it can be easily modified to produce a 192-bit or 256-bit key.

The average repeatability of the glitch count is 94.85% using this method. The overall distinguishability of the generated glitch counts for each output line is 10.46%.

Acknowledgments

I would like to thank my faculty advisor, Dr. Yong Kim, for guiding me through the many trials and tribulations that I have encountered throughout my research. I would like to thank 2d Lt Andrew Armstrong for the arduous task of teaching me how to use the Xilinx software programs. I would also like to thank 2d Lt Danielle Yaple for saving me hours of work by writing some code to organize my data. And finally I would like to thank 2d Lt Camdon Cady for his many insights into various aspects of my research.

Jennifer C. Anilao

Table of Contents

Page

List of Figures

Figure Page

List of Tables

Table Page

List of Acronyms

Abbreviation		Page
ASIC	Application-Specific Integrated Circuit…………………….	1
FPGA	Field Programmable Gate Array……………………….....	1
VHDL	VHSIC Hardware Description Language…………………..	2
AES	Advanced Encryption Standard…………………………….	2
ICs	Integrated Circuits………………………………………..	4
PUF	Physical Uncloneable Function.…………………………….	4
ID	Identity…………………………………………………...	6
LFSR	Linear Feedback Shift Register……………………………..	11
EDK	Embedded Development Kit………………………………..	22
SDK	Software Development Kit…………………………………	22
BRAM	Block Random Access Memory……………………………	30
UART	Universal Asynchronous Receiver/Transmitter………….….	30
SUT	System Under Test……………………………………….	31
CUT	Component Under Test…………………………………..	33

UTILIZING THE DIGITAL FINGERPRINT METHODOLOGY FOR SECURE KEY GENERATION

I. Introduction

1.1. Research Motivation

One way to implement user-specified hardware is to use a Field Programmable Gate Array (FPGA) because they avoid the upfront costs associated with Application-Specific Integrated Circuits (ASICs) [1]. However, the vulnerability of an FPGA to enemy attack is becoming apparent, as demonstrated by Standaert el al., in their results of power analysis attacks on FPGAs [2]. Previous research in this area has proven the validity of the digital fingerprint, but has fallen short in showing the practicality of this method. The motivation for coming up with a method to generate an encryption key using the unique, intrinsic characteristics of an FPGA is to provide more security for digitally stored data. To defeat the proposed method an adversary would need the user's password, the same FPGA used to generate the key, and the software program that was run on the FPGA.

1.2. Problem Statement

In order to generate a secure key, a random number or password must be provided to the encryption algorithm. Previous research by Capt Hiren Patel found that due to the random variations in the delays of transistors, FPGA's can be individually identifiable [3]. This "digital fingerprint" takes advantage of the slight variations in the characteristics of transistors, which introduces variations in the delay of the signal, causing glitches on the output of a gate or circuit.

By modifying the existing digital fingerprinting method, created by Crouch and improved by Patel, to produce either a zero or a one on each of its output lines, a user can specify which output line to use for each bit of the key. Since Patel's work typically produced glitches of between zero and eight on each output line using a 32-by-32 bit multiplier, a simpler combinational circuit is used for the intermediate circuitry for this application. This method investigates the use of a very simplistic NAND gate implementation with a programmable number of buffers added to the path to provide a greater variation in the number of glitches.

1.3. Goals

The first objective is to describe a combinational circuit in VHSIC Hardware Description Language (VHDL) that generates either none or one glitch. This will establish a binary decision, either a '0' or '1', occurring on the output of such a circuit. The zeros and ones should occur randomly on the output.

The second goal of this research effort is to generate a user-specified key to the Advanced Encryption Standard (AES) algorithm. Accomplishing the first goal will provide the user with the ability to choose whether each output line will produce a '0' or a '1' glitch count, enabling the user to specify a key.

1.4. Overview

The following chapters detail the investigation of using the digital fingerprinting method to generate a secure key for the AES encryption algorithm. Chapter 2 provides the background information on circuit identification techniques and the existing digital fingerprinting method. In Chapter 3, the methodology of this research is presented. This chapter also addresses the modifications made to the original Digital Fingerprinting method, and also some of the problems that were encountered and why they occurred. Chapter 4 discusses the results and the analysis of the data collected. Conclusions reached by this research effort are included in Chapter 5, along with possible ways to improve upon the key generation technique described herein.

II. Background

The following chapter presents background information relating to the generation of unique circuit identifiers. In order to comprehend circuit identification, it is essential to first understand why and how Integrated Circuits (ICs) of the same type can be distinguished from one another. The next section discusses the Physical Uncloneable Function (PUF) concept, and some of its implementations. The last section of this chapter covers the original Digital Fingerprinting method and the modifications to this method that are necessary to create an uncloneable key for an encryption algorithm.

2.1. Physical Variations Among Integrated Circuits

Although a particular IC may be fabricated using the same mask, slight dissimilarities will exist between each and every one of the circuits produced by that mask [4], [5]. The actual product of an IC manufacturing process is different from the idealized models we use to represent them due to the non-ideal methods by which the circuits are created. The idealized model of a transistor is represented in Figure 1. Figure 2 shows a picture of an actual transistor manufactured by Intel. Other research by Agarwal et al in [6] even suggests that variations in the characteristics are spatially correlated.

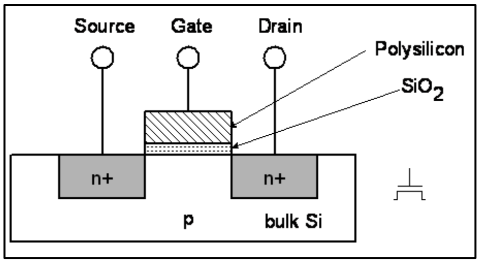


Figure 1: Ideal Model of a Transistor [3]

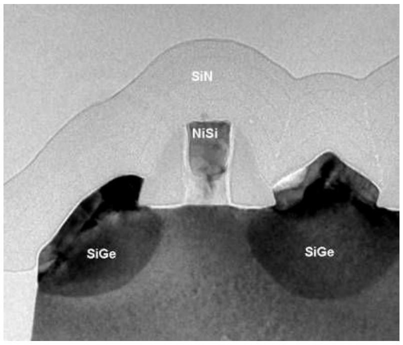


Figure 2: Actual Transistor [3]

There are many layers and junctions that compose an IC. Each process by which these layers and junctions are formed introduces the capacity for miniscule fluctuations to occur between actual devices, [7-9]. For example, ion implantation is used to form junctions in semiconductors. Ion implantation is a method of impurity doping where ions are accelerated toward a substrate through a patterned mask. The result is a doped region of scattered ions. Because the collisions are random, the final positions of the dopant atoms can only be estimated by a Gaussian distribution function [7]. There are also a small percentage of ions that tunnel through the silicon dioxide causing additional negligible disfigurements in the substrate.

Every process at every level of IC fabrication introduces a very small amount of error and the distribution of random allowable abnormalities that do not affect the overall function of the circuit. Researchers have taken advantage of these differences in developing efficient methods of circuit identification and encryption.

2.2. Physical Unclonable Functions

2.2.1. Overview

As technology continues to advance and become more and more a part of our daily life, so does the need for increased data security. Not only does the data need to be protected, but it also must be implemented in a way that is fast, reliable, small, and economical. With all of these limitations it is becoming increasingly difficult to provide this safeguard to consumers through conventional methods. Various researchers in [3-4, 10-13] have developed an innovative approach to Identity (ID) verification that takes advantage of the random physical variations in integrated circuits.

2.2.2. Arbiter PUF

2.2.1 Arbiter PUF Description

Figure 3 represents the arbiter PUF circuitry designed by Suh and Devadas [4]. The D latch (arbiter) determines the single bit output, Y, of the circuit based on signals that arrive on its inputs first. The delay paths that feed into the latch are comprised of pairs of multiplexers. Each i^{th} pair is controlled by X[i]. If X[i] = 0, the signal will continue on the same path. Otherwise, if X[i] = 1, the signals will exchange paths.

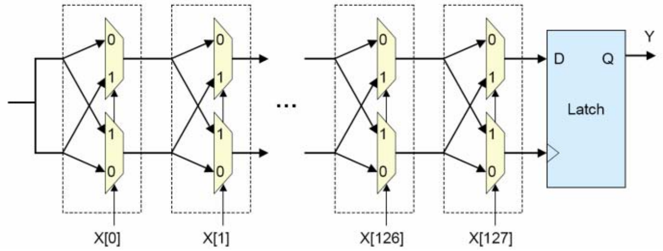


Figure 3: Arbiter PUF Circuit [4]

Because each multiplexer has a slightly different delay induced by the manufacturing process, the circuit will perform differently based on unique combinations of the input bits.

2.2.2 Arbiter PUF Reliability

The results of Suh and Devadas's experiments on the arbiter PUF circuit showed that when the arbiter circuit output was measured for the same chip, the output was only different 0.7% of the time, which suggests that a measurement based on the unique delays of a circuit is a valuable and reliable quantity [4].

There was a 23% chance of identical arbiter circuits on separate chips having different outputs given the same input. Suh and Devada pointed out that this low percentage was the result of not laying out their circuit symmetrically as it appears in the idealized illustration of Figure 3. If a truly symmetrical design is actualized, they claim that the inter-chip variation would approximately double.

The most reassuring aspect of the experiments conducted by Suh and Devadas is that the PUF circuitry is reliable when it comes to environmental stresses. The output noise stayed under 10% even when the circuit was faced with extreme operating temperatures of 100°C and a 33% voltage variance.

2.2.3. Ring Oscillator PUF

2.2.3.1 Ring Oscillator PUF Description

A ring oscillator PUF is shown in Figure 4 [4]. The circuit is composed of N oscillators, each of which oscillates at a different frequency due to the variations inherent in the manufacturing process as described in Section 2.2. The output of the circuit is a '1' if the frequency of the top signal is faster than the bottom frequency, and '0' otherwise. The output of these oscillators is fed into one of two multiplexers. The multiplexer selects frequencies to compare based on the control input.

The ring oscillator PUF is capable of producing up to $\log_2(N!)$ independent output bits.

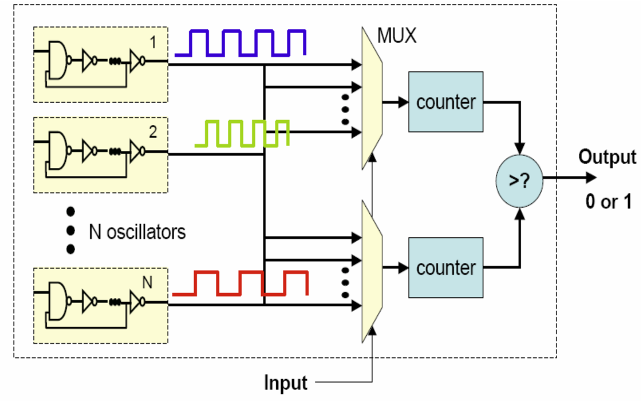


Figure 4: Ring Oscillator PUF Circuit [4]

2.2.3.2 Ring Oscillator PUF Reliability

It is important to ensure the oscillator comparisons are conducted between oscillators whose frequencies are not similar. These oscillators are very susceptible to changes in temperature, operating voltage, and other environmental factors. If an oscillator is subjected to extreme conditions and it is compared with an oscillator whose frequency is not far from its own, an erroneous output bit may result.

2.2.4. Butterfly PUF

2.2.4.1 Butterfly PUF Description

The details of the Butterfly PUF circuit are rendered in Figure 5 [5]. The Butterfly PUF is implemented in an FPGA by cross-coupling a pair of D latches. This configuration allows an input signal to excite the circuit, which in turn allows the circuit to transition from an unstable state to a stable state over time. This PUF derivative is for use on FPGAs, where the previous designs fall short. Upon excitation by an input signal, the circuit will enter an unstable state, where it will stay for a short period of time (on the order of picoseconds) before settling into a stable state. The circuit’s natural stable state may be a high or a low value depending on the slightly different delays on the connecting wires, brought about by the random variations experienced during the fabrication of the device.

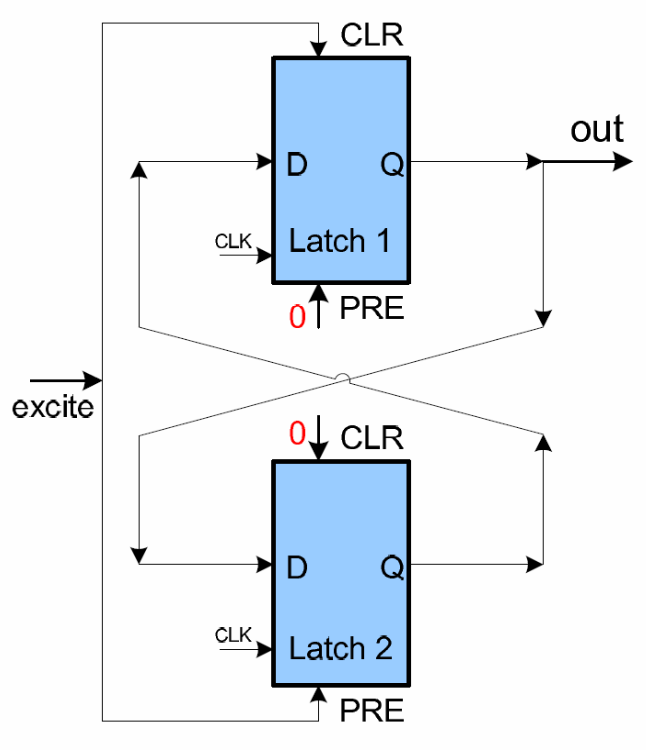


Figure 5: Butterfly PUF Circuit [5]

2.2.4.2 Butterfly PUF Reliability

Experiments conducted by Kumar et al. [5] deduce a clear separation between the within-class Hamming distance and the between-class Hamming distance. The within-class Hamming distance is the number of outputs that are different on the same device over multiple samples. The between-class Hamming distance is the number of outputs that are unique when comparing different devices. They also proved the ability to reproduce the same results under various environmental conditions. Among the conditions tested were temperature (-20°C to 80°C), operating voltage (not specified), and operating frequency (50 MHz to 120 MHz).

2.2.5. PUF Conclusion

The introduction of the PUF concept has revolutionized the realm of data security and cryptography. Varying levels of security have been accomplished using this single idea, depending on the particular implementation of the PUF. The full potential of PUFs has not yet been realized. Continuing research in this area is critical to developing this technology to its full potential. Future efforts should explore the ability to digitally fingerprint a standalone circuit and the effects of temperature, voltage, and radiation on that fingerprint. It is not yet known whether tampering can be sensed based solely upon the fingerprint of a circuit or not, although if confirmed, this could prove to be one of the most important ramifications that PUFs develop on matters related to national security.

2.3. Digital Fingerprint Background

The Digital Fingerprinting methodology was originally introduced by Crouch [14], and was improved by Patel [3]. The method is comprised of three main components, input generation, combinational circuit, and glitch capture.

Each of the 64 input bits of the combinational circuit is taken from a Linear Feedback Shift Register (LFSR). The seed value of the LFSR is hard coded in the glitch_counter_setup.vhd file. The LFSR produces a pseudo-random bit pattern that is fed directly to the inputs of the combinational circuit. The combinational circuit that is implemented in the design is a 32-by-32-bit multiplier. Each output of the multiplier is connected to the clock line of a 16-bit one-hot-state shift register, which captures the glitches produced by the multiplier. The one-hot-state shift registers are initialized to hold a ‘1’ in the least significant bit position. When a glitch appears, the ‘1’ shifts one position toward the most significant bit position. The final position of the ‘1’ in the register reveals the total number of glitches that occurred on the output line.

2.4. Cryptographic Algorithms [15]

2.4.1. Data Encryption Standard

DES was created as a result of a request to the general public, made by the United States government in 1976, for a standard cryptographic algorithm. The algorithm is a block cipher, which means that it partitions the message, or plaintext, into 64-bit blocks and encrypts each block before transmission. The key is a string of 64 bits; however, only 56 bits are used by the algorithm because every eighth bit is purely a parity bit.

The DES algorithm is executed by using three distinct stages, depicted in Figure 6. Stage one rearranges the initial bits of the message, m, into the initial permutation, m_0. The first 32 bits of m_0 is called L_0, and the second block of 32 bits is termed R_0.

Each round of the second stage uses a 48-bit string of bits from the key, called K_i. The following mathematical operations on rounds 1 through 16:

$$L_i = R_{i-1}$$
$$R_i = L_{i-1} \oplus f(R_{i-1}, K_i)$$

The function f expands and permutes R into a 4 x 12 matrix of 48 bit values, and stores it as $E(R)$. The result of $E(R)$ exclusive-OR K_i is split into eight 6-bit partitions that are each fed into their own S-box. The output of the S-box is a 4-digit binary number whose ith iteration is C_i. The C_i values are permuted once more to obtain the 32-bit result of the function f.

The final stage of DES swaps the position of L_{16} and R_{16} and applies the inverse of the initial permutation, which results in the ciphertext, *c*.

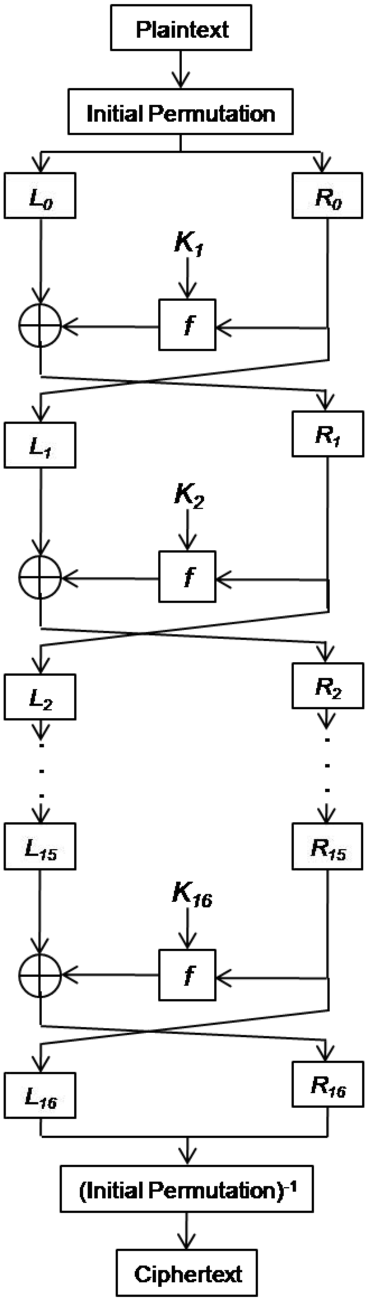


Figure 6: DES Algorithm

2.4.2. Advanced Encryption Standard

AES was also the result of a government call to the public for a standard, but this time it was for an algorithm capable of utilizing keys of 128, 192, and 256 bits. There are ten rounds of four steps, or layers, which compose the AES algorithm. The initial input bits to the algorithm are divided into a 4 x 4 matrix of 16 distinct bytes. A byte is 8 bits.

The first layer is the ByteSub Transformation. This layer permutes the input bits by changing their position in what is called an S-box. The output is a 4 x 4 matrix.

The second layer, the ShiftRow Transformation, keeps the first row of the input matrix the same. The second row of the matrix is circularly shifted left by one position. The third row is shifted by two, and the fourth row is shifted by three positions, in a circular manner.

MixColumn Transformation is the third layer. Each column of the output of the previous step is multiplied by a known, invertible matrix, shown in Figure 8 on the following page.

The AddRoundKey step is the final layer. This layer does an exclusive-OR operation on the round key and the output of the MixColumn Transformation step. The initial key is broken up into bytes and distributed into a 4 x 4 matrix, where each column is named *W(0), W(1), W(2),* and *W(3),* respectively. Each round expands the original matrix by four more columns. The round key for the *i*th round is a 4 x 4 matrix with the columns *W(4i), W(4i + 1), W(4i + 2),* and *W(4i + 3).*

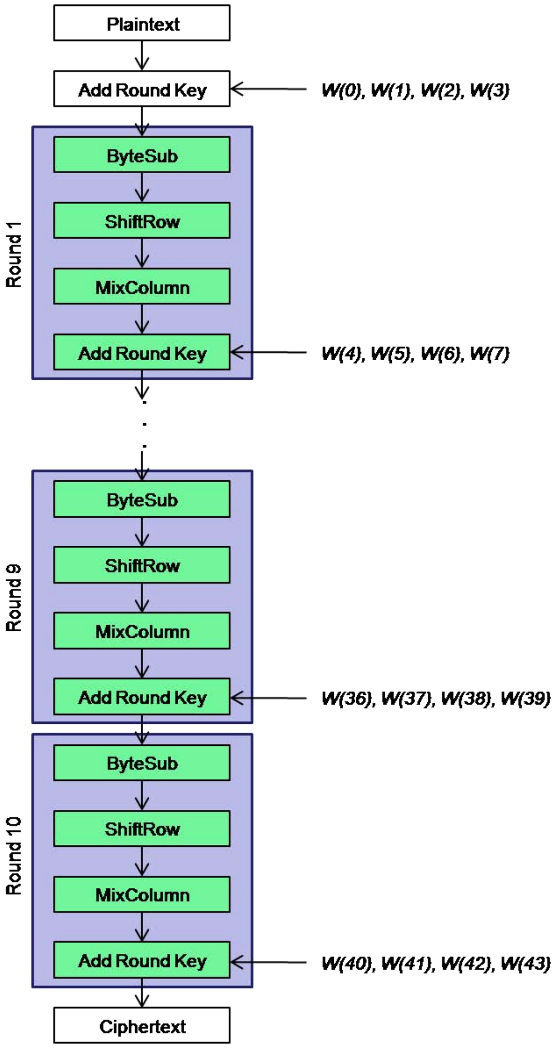


Figure 7: AES Algorithm

$$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix}$$

Figure 8: MixColumn Transformation Matrix

2.5. Background Summary

The variance of transistor characteristics in IC’s was presented in this chapter, as it relates to generating glitches. The PUF concept and some of its implementations were examined. Finally, the Digital Fingerprint Method, designed by Capt Patel, was outlined.

III. Methodology

3.1. Goals and Hypothesis

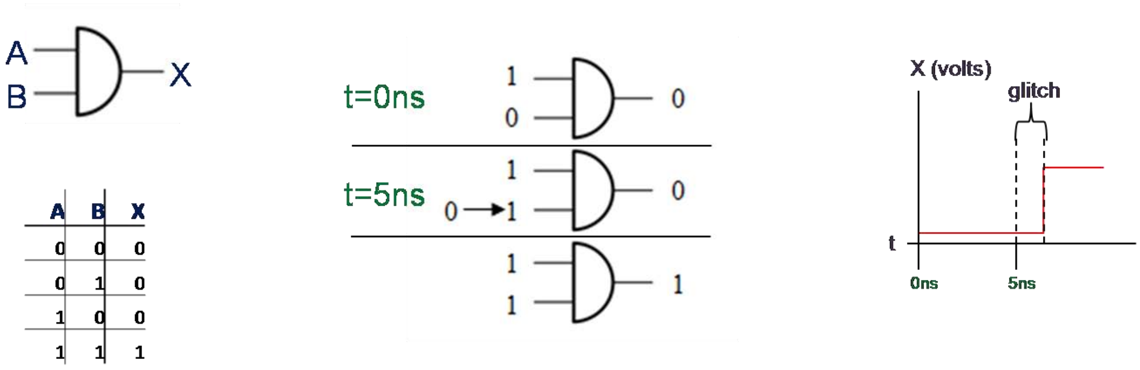
The first goal of this research is to design a circuit that will generate a stable glitch count on each output line of either ‘0’ or ‘1’ on each iteration. This research will build upon the existing digital fingerprinting method implemented by Patel. The intermediate circuit is to be modified in such a way as to induce either no glitches or one glitch on each output line. Theoretically, the best way to do this is to describe a very simplistic circuit in VHDL that consists of NAND gates with two levels of logic. In order to stimulate a glitch to appear on the output, a selectable amount of buffers should be added in between the two levels of logic. Logically, the more buffers that are added to the line, the higher the glitch count should be on the output.

Once the glitch count on each output line is either a zero or a one, and therefore a binary decision, the second goal is to make the buffer count of each intermediate circuit (each corresponding to one output line) selectable by a user. This enables the user to generate a key that is unique to the FPGA on which it is being generated. The user-specified key will then be used to encrypt plaintext using AES encryption. This ensures that an adversary would have to have the key, the exact FPGA that generated the original key, and the software used to generate and capture the key to be able to reproduce the same key.

3.2. Approach

3.2.1. Theory

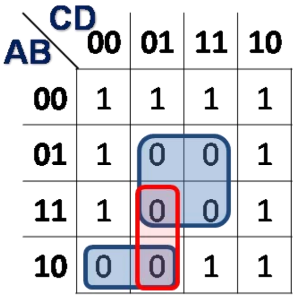
The first step in producing an uncloneable key is to come up with a way to produce between zero and one glitch on an output line. A glitch can occur on the output of a logic gate when the inputs to the gate cause its output to change. The output does not change immediately; it requires a small delay, on the order of picoseconds, to process the change, depending on the unique delay characteristics of the transistors used in the logic gate. The momentary false value appearing on the output when the inputs cause the output to change is called a glitch. A simple example of a glitch is illustrated in Figure 9. The AND gate on the top is the initial condition with inputs 0 and 1 and an output of 0. The middle gate shows how the output of the AND gate is still 0 after the second input changes to a 1 at 5ns. The bottom AND gate shows the correct output after it has had time to process the change in inputs.



A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

Figure 9: Simple Example of a Glitch Occurring on an AND Gate

These glitches will make up the bits of the key used in an encryption algorithm. To simplify the glitch generation concept used in the original digital fingerprinting method, it makes sense to make the glitch circuit as simple as possible. The proposed idea is a two level logic circuit using three input terms. The Karnaugh map is shown in Figure 10, and the circuit is illustrated in Figure 11. It is theorized that adding buffers to one of the input lines of the gate directly before the output will increase the delay and eventually cause a glitch to appear on the output.



AB \ CD	00	01	11	10
00	1	1	1	1
01	1	0	0	1
11	1	0	0	1
10	0	0	1	1

Figure 10: Karnaugh Map for a Simple Glitch Circuit

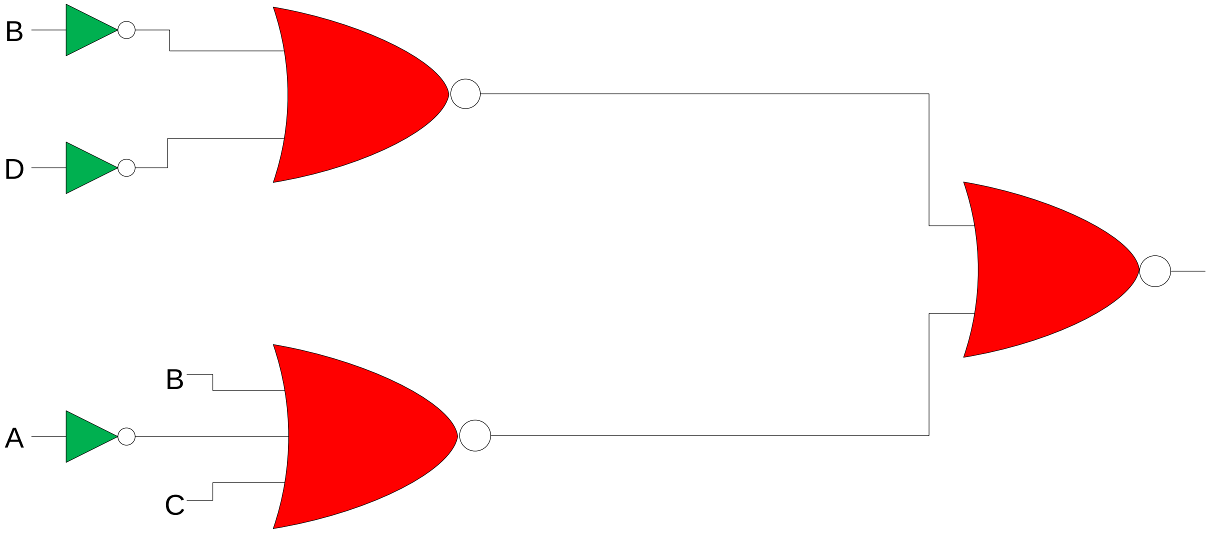


Figure 11: Simple Glitch Circuit

3.2.2. Simulation

The next step is to simulate the concept to approximate the number of buffers needed to cause a delay long enough to produce a glitch. This step is carried out to get an idea of how many buffers will need to be added to the path. The results of simulating the design will not likely yield a very precise solution, as a lot of researchers have studied glitch modeling with commercial software products and have not found an accepted method of achieving an accurate simulation, [16-19]. The buffers are added directly before one of the input lines to the last NOR gate before the output. An input combination that switches from one maxterm combination to another is used to stimulate glitches. The maxterms of a circuit are the combination of inputs that cause the circuit's output to equal '0'. For example, in Figure 10, the maxterms that are highlighted in blue are A'+B+C and B'+D'. The first simulation run shows that a 1.0311ns glitch can be seen at the output of a four input simple logic circuit, see Figure 12. This pulse width is very short and would not likely be detected in actual circuitry due to the setup time of the flip flops.

The next simulations show the pulse width increasing as the number of buffers increase. The addition of four buffers, shown in Figure 13, causes a glitch pulse width of 3.2260ns, which should be a sufficient amount of time to be able to sense it. Simulations revealed that adding four buffers into the circuit should introduce enough delay into the circuit to capture a glitch.

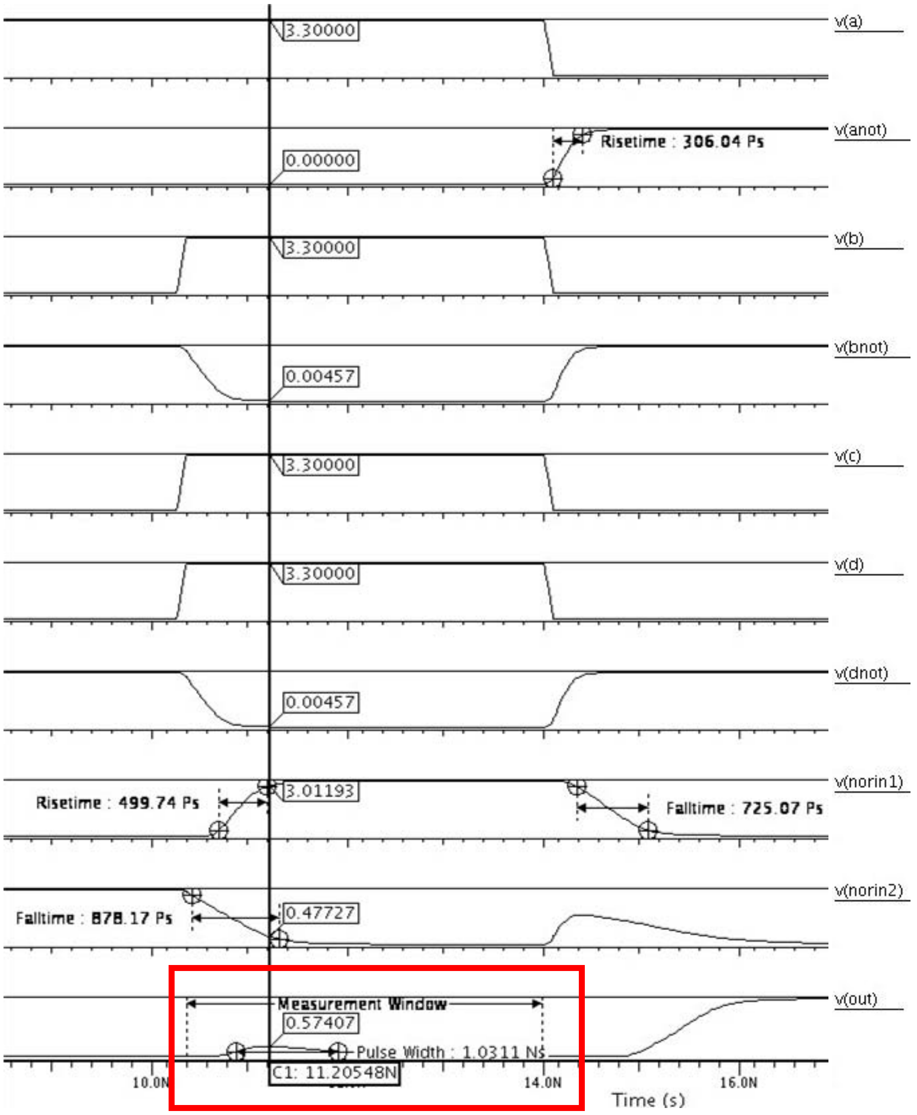


Figure 12: Glitch Circuit Simulation with Zero Buffers

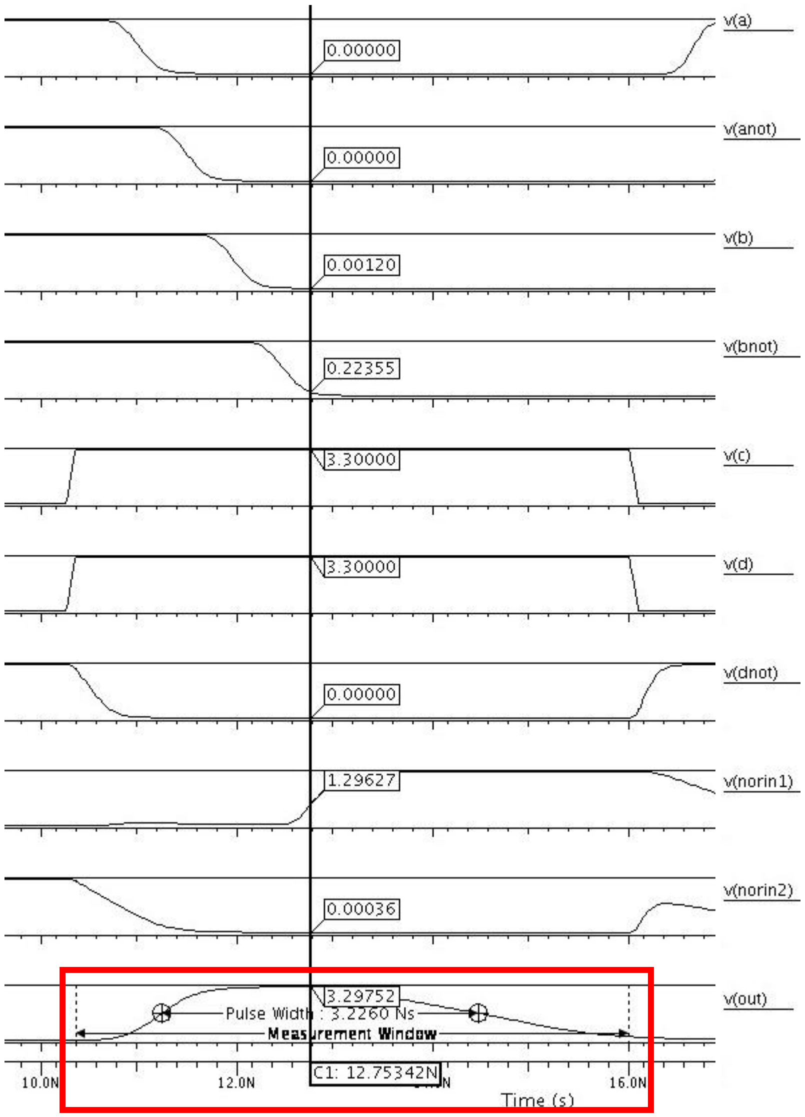


Figure 13: Glitch Circuit Simulation with Four Buffers

3.2.3. Implementation

The implementation of the circuit is the next logical step in the progression. The design is implemented on a Xilinx Virtex-5 FPGA, using Xilinx Embedded Development Kit (EDK) as the development tool. The glitch calculations and user applications are built using Xilinx Software Development Kit (SDK). The flow chart depicted in Figure 14 illustrates the steps required to accomplish the end goal of generating a user-specified key to use in an encryption algorithm. First, an encryption scheme must be chosen. Next, a circuit that generates glitches must be designed. After this, there must be written in C that uses the Power PC on the Virtex-5 to characterize the number of glitches for each buffer count. This information is used to generate the key in the last step.

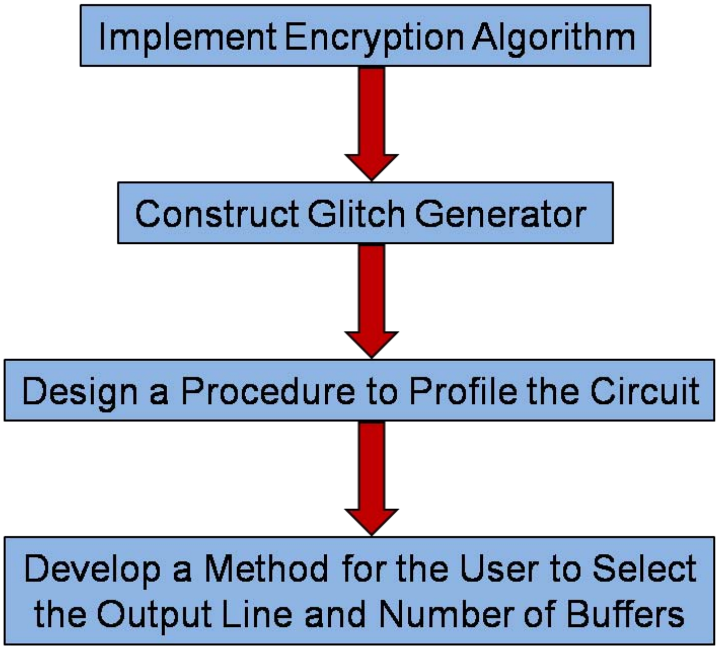

Implement Encryption Algorithm
Construct Glitch Generator
Design a Procedure to Profile the Circuit
Develop a Method for the User to Select the Output Line and Number of Buffers

Figure 14: Implementation Flow Chart

3.2.3.1 Implement Encryption Algorithm

The first step in implementation is to employ the encryption algorithm. The National Security Administration has approved AES as its cryptography standard. AES is capable of using key sizes of 128, 192, and 256 bits [20]. The Committee on National Security Systems authorizes the use of 128 bits for SECRET information [21]. TOP SECRET information requires key lengths of 192 and 256 bits. The AES encryption modules, written in VHDL, are provided by a fellow AFIT student, Major William Cobb.

3.2.3.2 Construct a Glitch Generator

The next step is to construct a circuit that generates between zero and one glitches for each output line. The first task in realizing a glitch generator is to make modifications to Capt Patel's digital fingerprinting code. The first of these modifications is a new intermediate circuit that replaces the multiplier. The new circuit is capable of generating zero or one glitch on each output line. This glitch circuit is modified from the one used in the simulations. It has three input terms (A, B, and C), and it uses NAND gates instead of NOR gates. One less input term is used to reduce the complexity of the circuit. NAND gates are utilized so that the glitch capture code does not need to be modified since a NAND circuit will generate 0's as glitches, while a NOR implementation will generate 1's as glitches. Although the circuit is different from the one used in the simulations, for the current research purposes, it is essentially the same. Both the NAND and NOR implementations are very simple circuits that are capable of generating glitches. The simulations are only accomplished as an approximation of how many buffers it would take to produce a glitch wide enough that it could be captured by a one-hot state shift register. Since both circuits use two levels of simple logic gates, and both use a similar number of transistors, the delay that buffers introduce into the design should be about the same.

A regular shift register is used in place of the LFSR to make the inputs to the circuit known and stable rather than pseudo-random. The input combination must change only once to ensure that no more than one glitch will be induced.

Another modification is the utilization of slave registers in the user logic file for the inputs to the shift register. This allows the inputs to be controlled directly from the C code, permitting any changes to be made rapidly instead of having to wait for the entire design to re-synthesize. Since the design uses a 64-bit shift register and the slave registers are only 32 bits wide, the new design uses 32 bits that are hard-coded into the VHDL and the other 32 bits come from the C code. It is not imperative that all 64 bits come from the C code. Having the ability to change 32 bits of the input provides a sufficient amount of flexibility to be able to adapt to any new requirements that may arise in the future. It also provides a longer sequence of the input combination to repeat, which allows the circuit time to settle to the correct value if the glitch takes longer than one clock cycle to propagate through the circuitry.

One problem that is present in the original design is that the set and reset combination that is given to the input shift registers and the one-hot state shift registers. The shift register needs the sequence 0, 1, 1 on the set lines, and the reset sequence needed is 0, 0, 1. The one hot state shift register needs the reset sequence 0, 1. This causes all registers to go to zero, except the least significant bit. To fix this, ‘set’ is now connected to the ‘clear’ input and ‘reset’ is connected to the ‘load’ input of the shift register.

The C code, written by Capt Patel, is hard to decipher because is lengthy and hard to follow. To make the code more readable, there are new functions in the code to make the program flow easier to understand, including printDF() and setInputSequence(). The printDF() function prints the glitch count for each of the 64 output lines of the five glitch units. It also displays the glitch total for each of the five units. The setInputSequence() function is called at the beginning of the program to provide the proper sequence of values to the input shift registers, which hold the inputs A, B, and C to the glitch generating circuits. Theoretically more functions could be added to make the code more readable, but the time between setting the values and reading them are critical in a circuit that relies on momentary false values. For example, a function to provide the proper set and reset sequence does not result in the same glitch counts on all of the output lines because the values change by the time the function returns.

When running the original main.c file, the glitch count for each output is either ‘0’ or ‘2’. This is not the expected outcome. Since the input combination should only produce an outcome of ‘1’ and only changes once, the only outcomes should be a ‘0’ or a ‘1’. Upon further investigation, the error is in the portion of the C code that counts the glitches by shifting the bits of the one-hot state shift register. The code shifts the bits of the 16-bit counter until the left-most ‘1’ reaches the 16th bit position. The original code subtracts the number of shifts it took to get the bit to the 16th position from 16. This is incorrect because if the glitch count is ‘1’, that bit originates in the 2nd bit position. The code will shift the bit 14 times to reach the 16th position, and 16-14 = 2. The code is now modified to subtract the number of shifts from 15. Returning to the previous example and using the updated value to subtract from, 15-14 = 1, which is the correct interpretation of the glitch count.

After all of the initial modifications were made to the digital fingerprinting code, several other changes were needed to correct Xilinx-specific issues. In the earliest renditions of the Uncloneable Key Generation System, no glitches show up on the output lines. After experimenting with several possible solutions, it is apparent that Xilinx optimizes the design by removing the buffers. This can be determined by examining the Device Utilization Summary in the system_xst.srp file. Since the number of buffers is doubled while the amount of resources utilized by the FPGA stays the same, it is clear that the design is being optimized by removing unnecessary gates. Xilinx considers these buffers unnecessary circuitry because buffers do not change the final outcome of the circuit. For the purposes of this research, the buffers are used to cause an increase in the propagation delay from the input to the output of the simple glitch circuit. To keep Xilinx from removing the buffers, attributes DONT_OPTIMIZE and KEEP_HIERARCHY are included in the VHDL file that describes the buffer and the glitch circuit file that multiplexes buffers together [22].

Another issue inherent in the VHDL code was discovered when glitch counts on all outputs were 1's. At this time, buffers were only added to the path of the top input to the second level NAND gate. The problem arises due to the difference in the delay induced by the multiplexed buffer unit as compared to a wire. Even with no buffers added to the path, the delay is long due to the multiplexers on one wire. At the time of the discovery, there were nine multiplexers on each path, which adds a significant amount of delay to each output line. This issue is resolved by adding the capability to add buffers to both the top and bottom input of the second level NAND gate as depicted in Figure 15 for the linear selection implementation and in Figure 16 for the cascaded buffer design.

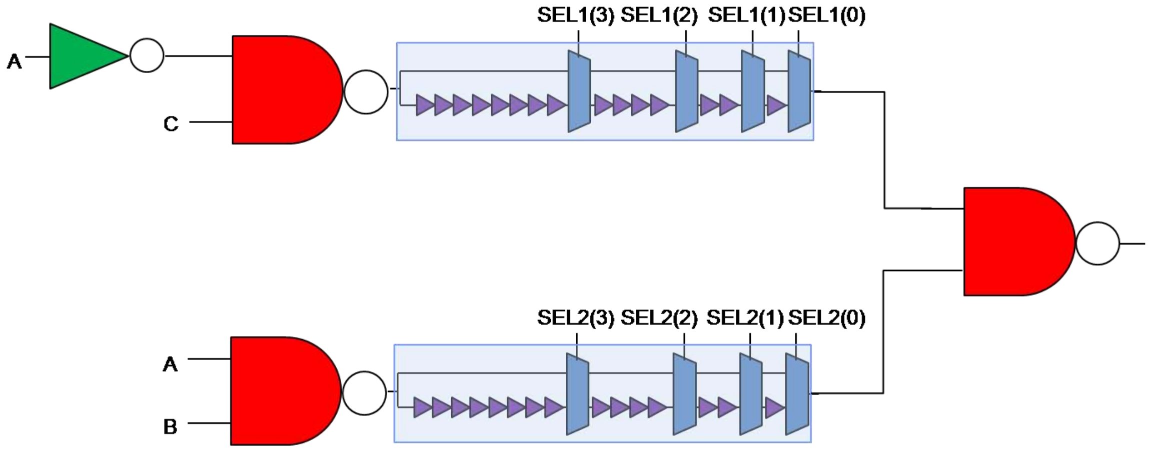

SEL1(3) SEL1(2) SEL1(1) SEL1(0)
A
C
SEL2(3) SEL2(2) SEL2(1) SEL2(0)
A
B

Figure 15: Glitch Circuit with Linear Buffer Selection on Both Inputs to NAND Gate

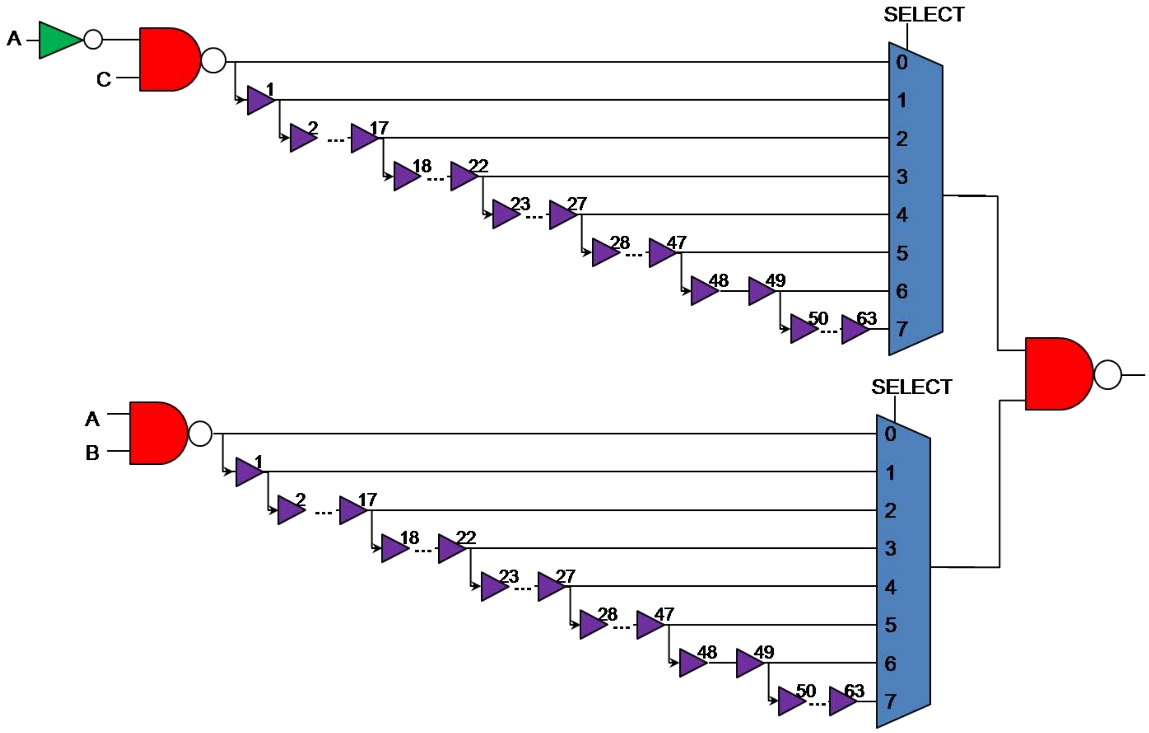


Figure 16: Glitch Circuit with Cascaded Buffer Selection on Both Inputs

Another realization that an initial assumption was incorrect occurred when glitch counts did not strictly increase or decrease when increasing the number of buffers on a circuit path. Instead, glitch counts are random when buffer counts on one path are increased. This is a result of how the components are wired together. Xilinx EDK assigns random wiring paths between all components in the design. Although, theoretically, two components are next to each other, Xilinx may not put an efficient, short, straight path between them. It may take an inefficient, long, winding path to the next component. The placement of components and wiring paths are random. After analyzing the data, it could be ascertained exactly which paths were causing delays in the output. These paths were ones both with and without buffers between multiplexers.

The location and/or content of some xil_printf statements cause the program to halt execution prematurely, print random ASCII characters, or fail to begin execution. Similar problems can occur when removing an unused variable or not initializing the unused variable.

3.2.3.3 Design a Procedure to Profile the Circuit

The third step in the implementation flowchart is to create a procedure to profile the circuit. Profiling the circuit is simply generating glitch counts for all quantities of buffers and recording the results. The data collected during profiling is for the user to determine if an output line is usable, and what glitch count is achieved by inserting a particular number of buffers into a glitch circuit path. An output line is considered unusable or "bad" if the same glitch count is achieved for all quantities of buffers inserted into the upper and lower input of the final NAND gate of the glitch generation circuit. Several issues materialized while working on this part of the application.

It is a known fact that temperature affects transistor characteristics. The temperature of the surrounding environment is a macroscopic example of heat that affects the circuitry. For this research, all measurements were taken at room temperature to ensure that the ambient air temperature did not affect the output. Another, less obvious issue involving heat is the effect of executing the C code. Running the C code causes currents to travel through the FPGA, which generates heat within the circuit. The glitch counts tend to be different the first time the code is executed after a period of rest. To stabilize the glitch count, the glitch counting portion of the code is run through five iterations before displaying the glitch counts. This is controlled by a for loop with the variable “itr”, which is short for iteration.

Another repercussion of heat in the design deals with the print statements. Changing the location of xil_printf statements in main.c causes the glitch counts to vary. The glitch units that are placed close to the Block Random Access Memory (BRAM) and Universal Asynchronous Receiver/Transmitter (UART) interfaces stay hotter, while the units that are farther from those components cool off while print statements execute. Print statements are necessary, and they cannot always wait until after the code is finished running, therefore the solution is to be cognizant of the issue. The full C code must not be changed in between profiling the circuit and generating the key. Also, the print statements within the portions of code that initiate glitches and read the glitch count must be the same in both the profile and key generation methods to avoid getting contrasting glitch counts.

3.2.3.4 Develop a Method for the User to Select a Location of the Bits of the Key

The last step of implementation is to generate the key. The result of profiling the circuit in the previous step is used to generate the correct key. The data collected reports the glitch count for each output line, for every quantity of buffer available. The user can then determine which outputs not to use, as well as the buffer count and output to choose for each bit of the key they would like to generate. For example, if a user wanted to generate '1', they would look at the profiling results and find the buffer quantity on a usable output line of a particular circuit that gave a glitch count of '1'. These three pieces of information are entered into the key generation portion of the code for each bit of the 128-bit AES encryption key. The key generation code is executed in C, and is very similar to the profile portion of the C code, except that there is a user entry loop that asks the user for a circuit number, an output line, and the number of buffers to insert into the glitch circuit.

3.3. System Boundaries

The System Under Test (SUT), illustrated in Figure 17, is the Uncloneable Key Generation System, consisting of three 64-bit shift registers, intermediate circuits, and 64 one-hot state shift registers. Each of the 64-bit wide shift registers serves as an input line of the intermediate circuitry. There are three shift registers for each glitch circuit, each holds an input (A, B, or C) that is fed to the glitch circuit. They provide an input pattern that shifts one bit into the circuit on every clock cycle. There are 64 intermediate circuits. Each one generates glitches for a single output line. The glitches are recorded by the 16-bit one-hot state shift registers.

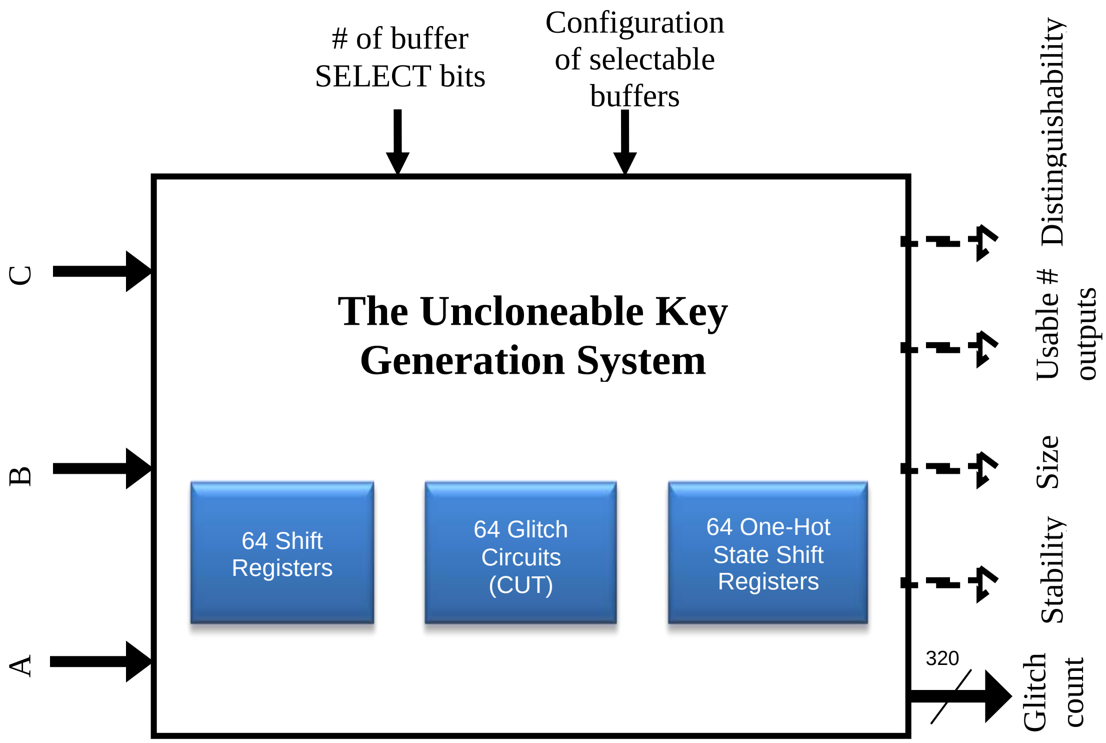


Figure 17: System Under Test (SUT)

A 16-bit one-hot state shift register is a register that contains all zeros except for one register, which is the "hot", or active, state. The clock line of each one-hot shift register is tied to an output line of the combinational logic circuit. Each time a glitch occurs, it moves one position over in the register. This concept is illustrated in Figure 18. The final position of the one in the register reveals the total number of glitches that occurred on the output line.

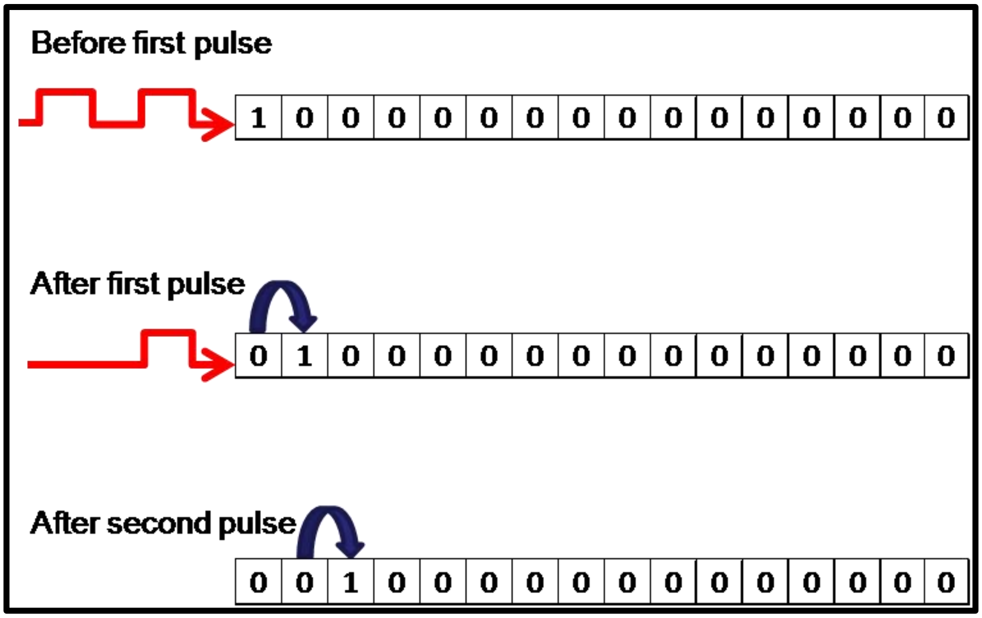


Figure 18: 16-bit One-hot State Shift Register

The Component Under Test (CUT) is the combinational circuit that generates glitches on the output lines. This component is comprised of 64 independent circuits. Each one is a 2-level NAND circuit that combines two minterms. Minterms are similar to maxterms, explained in section 3.2.2, except that minterms are the combinations of inputs that cause the output to evaluate to a '1'. The output of the first level is connected to a selectable number of buffers. These buffers are inserted to induce glitches to appear on the output lines. The output of the buffers is wired to the second level NAND gate, whose output goes to the clock line of a one-hot state shift register to capture the glitches produced by the previous component. The exact configuration of the buffers in the intermediate circuitry is assessed in this research.

3.4. System Services

The Digital Fingerprinting System provides a unique identifier for an electronic circuit. The system must be able to produce varying glitch counts when adding different numbers of buffers into the circuit path. The output of the system is composed of five glitch units, each with 64 output lines, for a total of (64 x 5) = 320 possible output lines. The user can select 128 of the outputs to use as a key input to the AES encryption algorithm.

In previous experiments [3], the distinguishability and stability are evaluated. The distinguishability is the measure of how well the digital fingerprinting method is able to generate a unique glitch count for a large amount of electronic circuits. The stability is a measure of the method's ability to reproduce the same glitch count for the same circuit on multiple runs. Given that the Digital Fingerprint System has already proven to deliver a unique glitch count for different FPGAs, the system should not have a problem with distinguishability. Stability, on the other hand, is more of a concern. The stability of the glitch count is the primary service that the system provides.

3.5. Workload

The workload of the Digital Fingerprinting System is the input bit combination that is loaded into the shift registers initially. The NAND glitch circuit produces a '1' on its output on minterms one, three, six, and seven. The current implementation begins with minterm seven, followed by minterm three. This input combination will cause some outputs to produce glitches as it switches from one input to the next.

3.6. Performance Metrics

3.6.1. Size of the circuit

Keeping the size of the circuit as small as possible is important. The end application of this device will make it infeasible if it is too large. The original digital fingerprinting code used almost the entire design space on a Xilinx Virtex-II Pro. The smaller the footprint of the key generation hardware, the more useful it will be to a user.

3.6.2. Number of usable output lines

The number of usable output lines is an important measure to take into account. If the design is to be kept as small as possible, there should be as few wasted lines as possible. A usable line is one that has at least one '0' and one '1' glitch for all possible numbers of buffers. If an output line doesn't have at least one of each glitch count, it is considered unusable because there is no way for the user to specify the output on that line.

3.6.3. Stability

The internal stability of the output line glitch counts, as outlined by Patel in [3], is ascertained by taking multiple readings of the glitch count on each output line and determining how many times the glitch count varied over those readings. Each time the DF algorithm is initiated, each output line is sampled 100 times in a row. A running total of the glitch count is kept for each output line. If the total glitch count is not 0 or 100, that output line is not completely stable. Since keys must be stable, any buffer count that causes an output line to be less than 100% stable should not be used.

3.6.4. Distinguishability

The distinguishability of a circuit that is used to generate an uncloneable key should be very high. In this case, distinguishability determines the likelihood of an ID collision. Suh describes the probability of an ID collision in [4]. An ID with X bits has the possibility of producing 2^X distinct IDs. This modified DF method contains five DF units, each with 64 output lines. The total number of available outputs is (64 x 5) = 320, so the number of possible IDs is 2.14 x 10^{96}. In the following equation, Y represents the total population of chips, *n* represents the *nth* chip, and X is the number of bits in the ID.

$$P_{collision} = 1 - \prod_{n=1}^{Y} \left(1 - \frac{n-1}{2^X}\right) \qquad (1)$$

If Y is set to 1,000,000 and X is 2.14 x 10^{96}, the theoretical possibility of a collision is virtually zero. The modified DF algorithm will be run on five Virtex-5 boards to establish distinguishability among a small sample of circuits.

3.7. System Parameters

3.7.1. Number of buffer SELECT bits

The number of buffer select bits determines the total number of buffers that can be inserted into the path on each output line. The linear selection implementation is capable of inserting up to 2^B buffers, when B is equal to the number SELECT bits. The cascading design is theoretically capable of placing any number or buffers on the path, since the only requirement is that as the SELECT bit combination increases, so does the number of buffers. If B is the number of SELECT bits, the design will have 2^B different choices for the number of buffers inserted. The number of buffers in between each cascaded level can be any number. The design implemented in this research allows the user to choose between eight different buffer amounts depicted in Table 2 (0, 1, 17, 22, 27, 47, 49, and 63), using three SELECT bits.

3.7.2. Configuration of selectable buffers

There are a multitude of different possible ways to configure the buffers in a way that makes them selectable to the user. This research tests two configurations for adding a selectable number of buffers to the circuit to induce glitch generation. The first design allows the user to select the number of buffers by using binary bits. A multiplexer for each bit selects whether the buffers between it and the multiplexer before it are used. An illustration of this concept using four buffer SELECT bits can be seen in Figure 19.

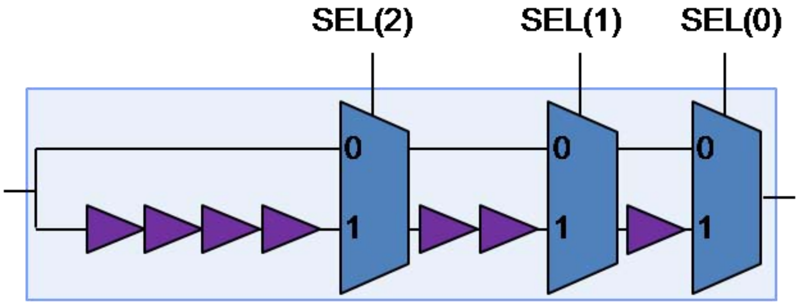


Figure 19: Linear Buffer Selection Using a Multiplexer for Each Binary Bit

The second design tested, shown in Figure 20, is one that cascades buffers together to insert different buffer totals into each line of one multiplexer. The cascading buffer selection design is an attempt to reduce the delay that is introduced by adding multiple multiplexers in series. The linear multiplexing circuit was analyzed to figure out what number of buffers produced the most usable output lines as described in Section

3.6.2. The results of the analysis of the first glitch unit are detailed in Table 1. The other four glitch units show similar results.

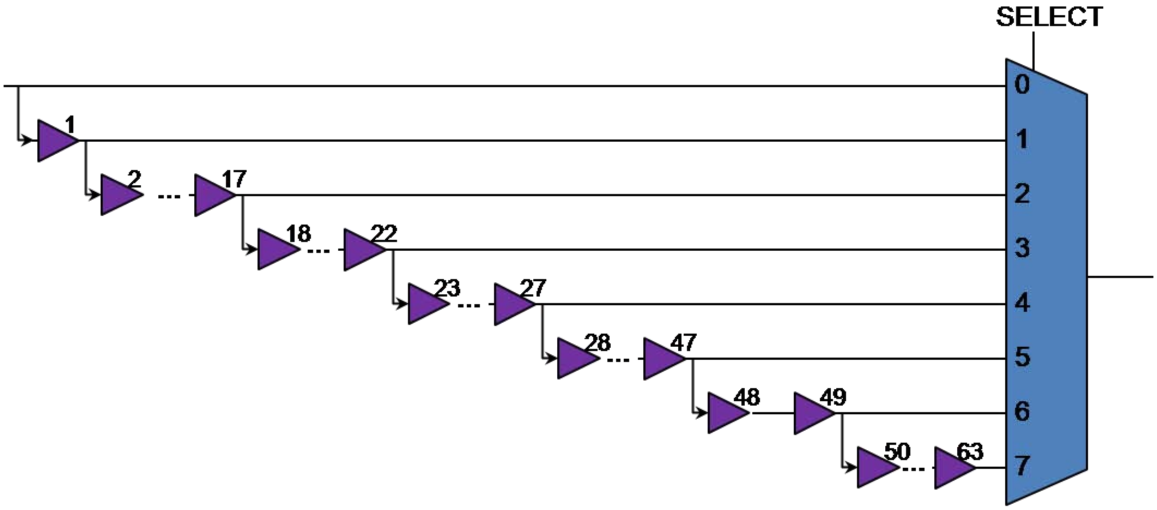


Figure 20: Buffer Selection Using a Multiplexer and Cascading Buffers

Table 1: Analysis of Usable Output Lines for Cascading Buffer Design

	Top Buffer Count:	0	1	22	47	63	0	0	0	0	# of:		T' if row has a 100 & a 0
	Bottom Buffer Count:	0	0	0	0	0	1	22	47	63	100's	0's	
Output Line #	0	0	0	100	100	100	100	100	100	100	7	2	T
	1	0	100	0	100	100	0	0	100	100	5	4	T
	2	0	0	0	0	0	100	100	100	100	4	5	T
	3	0	0	100	100	100	0	100	100	100	6	3	T
	4	100	100	0	100	100	100	100	100	100	8	1	T
	5	100	100	100	100	100	0	100	100	100	8	1	T
	6	100	100	100	0	100	100	100	100	100	8	1	T
	7	0	100	100	100	100	0	100	100	100	7	2	T
	8	100	100	100	100	100	100	0	0	100	7	2	T
	9	100	0	100	100	100	100	100	100	100	8	1	T
	10	0	100	100	100	100	0	100	100	100	7	2	T
	11	0	100	100	100	100	100	100	100	100	8	1	T
	12	0	0	100	18	100	100	100	100	100	6	2	T
	13	0	100	100	100	100	0	0	0	0	4	5	T
	14	100	100	100	100	100	100	0	0	0	6	3	T
	15	0	100	100	100	100	0	0	0	0	4	5	T
	16	0	0	100	100	100	0	0	0	0	3	6	T
	17	100	100	100	100	100	100	100	100	0	8	1	T
	18	0	0	0	100	100	0	0	0	0	2	7	T
	19	0	100	100	100	100	0	0	0	0	4	5	T
	20	0	100	100	100	100	0	0	0	0	4	5	T
	21	100	100	100	100	100	100	0	0	0	6	3	T
	22	0	100	100	100	100	0	0	0	0	4	5	T
	23	0	0	0	0	100	0	0	0	0	1	8	T
	24	0	0	100	100	100	0	0	0	0	3	6	T
	25	0	0	100	100	100	0	0	0	0	3	6	T
	26	0	0	100	100	100	0	0	0	0	3	6	T
	27	0	0	100	100	100	0	0	0	0	3	6	T
	28	0	100	100	100	100	0	0	0	0	4	5	T
	29	0	100	100	100	100	0	0	0	0	4	5	T
	30	0	100	100	100	100	0	0	0	0	4	5	T
	31	100	100	100	100	100	0	0	0	0	5	4	T
	32	0	100	100	100	100	100	100	100	100	8	1	T
	33	100	100	100	100	100	0	100	100	100	8	1	T
	34	0	0	100	100	100	0	100	100	100	6	3	T
	35	100	100	100	100	100	0	0	0	100	6	3	T
	36	0	0	0	0	100	0	0	0	100	2	7	T
	37	100	0	0	100	100	100	100	100	100	7	2	T
	38	100	100	0	100	100	100	100	100	100	8	1	T
	39	100	100	0	100	100	100	100	100	100	8	1	T
	40	100	100	0	0	0	100	100	100	100	6	3	T
	41	100	100	100	100	100	0	0	100	100	7	2	T
	42	100	100	100	100	100	0	0	100	100	7	2	T
	43	0	0	0	100	100	100	100	100	100	6	3	T
	44	100	0	0	0	0	100	100	100	100	5	4	T
	45	0	0	0	0	0	0	0	100	100	2	7	T
	46	0	0	0	0	0	0	100	100	100	3	6	T
	47	100	100	100	0	100	100	100	100	100	8	1	T
	48	100	0	0	0	0	100	100	100	100	5	4	T
	49	0	0	0	0	0	0	100	0	100	2	7	T
	50	0	0	0	0	0	100	100	100	100	4	5	T
	51	100	0	0	0	0	100	100	100	100	5	4	T
	52	100	100	0	0	0	100	100	100	100	6	3	T
	53	0	0	0	0	0	0	0	0	100	1	8	T
	54	100	0	0	0	0	100	100	100	100	5	4	T
	55	0	0	0	0	0	0	0	100	100	2	7	T
	56	100	100	100	0	0	100	100	100	100	7	2	T
	57	100	100	0	0	0	100	100	100	100	6	3	T
	58	0	0	0	0	0	0	100	100	100	3	6	T
	59	0	0	0	0	0	0	0	0	100	1	8	T

3.8. Factors

3.8.1. Number of buffer select bits

The number of bits to select the amount of buffers used will be minimized to ensure that the entire design is as small as possible. The design is implemented with 3, 4, and 6 buffer SELECT bits. When using the multiplexed linear selection implementation, the maximum number of buffers on each line is equal to 2^{B-1}, where B represents the number of SELECT bits. Maximum buffer counts of 7, 15, and 63 are examined when studying the linear selection implementation. Each design is capable of choosing between zero and the maximum number of buffers since it uses binary selection.

The cascading multiplexer uses three selection bits to choose eight different buffer counts. The buffer counts for this design are shown in Table 2 below.

Table 2: Number of Buffers Chosen According to SELECT bits

SELECT bits			# of
2	**1**	**0**	**buffers**
0	0	0	0
0	0	1	1
0	1	0	17
0	1	1	22
1	0	0	27
1	0	1	47
1	1	0	49
1	1	1	63

3.8.2. Configuration of buffers

The configuration of the buffers is investigated to determine the most effective method to generate at least one of each of the two main glitch counts, ‘0’ or ‘1’. One configuration is the linear selection implementation, where each bit position of the SELECT line corresponds to the number of buffers that is placed in the circuit.

The other circuit is designed to use the previous input to the multiplexer, with a specific number of buffers added to the path into the multiplexer. This is useful if certain, nonlinear buffer counts prove to be the best way to increase the delay on a path.

3.9. Evaluation Technique

The Digital Fingerprinting System will be evaluated via the measurement of a real system. Some preliminary studies will use simulation to deduce an appropriate starting point, but the vast majority of experiments will require measuring the outcome of the actual system. Results are taken using the linear selection design with 7, 15, and 63 maximum specifiable buffers on each line, and from eight different buffer counts (0, 1, 17, 22, 27, 47, 49, and 63).

The experimental configuration consists of 64, 64-bit shift registers for each of the three inputs to the circuit (A, B, and C), 64 simple combinational logic circuits, and 64 16-bit one-hot state shift registers to record the glitches on each output line.

The inputs A, B, and C will use minterm seven followed by minterm three. Each minterm is repeated 32 times to fill up the shift register. The shift registers are kept 64 bits wide so that the design could easily be adapted to produce more glitches, it gives the output adequate time to change, and it is simpler to leave the design the way it is.

The combinational logic circuit is a simple 2-level logic circuit that combines two terms. The main circuit uses three NAND gates and an inverter. The selectable buffer units use buffers and multiplexer(s), and are placed before the second level NAND gate on both its top and bottom inputs.

Each glitch circuit is connected to the clock line of a one-hot state shift register. The most significant bit of each of the registers is initially set to ‘1’, signifying the hot state. Each glitch acts like a clock pulse and causes the hot state to move one position toward the least significant bit. At the end of the simulation, the position of the one in the shift register determines how many glitches occurred.

3.10. Experimental Design

The number of simulations run to determine the simulation method has been successful is based on experiments and justification by Patel [3] since a full factorial, or even partial factorial number of simulations on several factors over the thousands of transistors in a multiplier circuit would be too time consuming. Patel uses the following equation to compute the confidence interval of his experimental method where $z_{\alpha/2}$ is 1.96, determined by a normal distribution table.

$$d = \pm z_{\alpha/2} [p(1-p)/n]^{1/2} \tag{2}$$

The p represents the probability estimate. If all of the circuits were able to be uniquely identified, p would be 100%, which cannot be used in the above equation; therefore p = (n-1)/n, where n is the sample size, which is chosen as 25 since there are five Virtex-5 boards available, and the project instantiates five complete glitch units. With these numbers, if each circuit has a unique ID, there is 95% confidence that 96% ±

7.68% of all circuits will have a unique identity when evaluated by this simulation method.

3.11. Methodology Summary

The number of buffer SELECT lines and configuration of buffers is varied to deduce the most reliable and efficient means of producing at least one ‘0’ and one ‘1’ on each output line. Once the best configuration of these factors and levels is determined, this project provides an interface that allows a user to profile the design to determine if/which output lines are unusable. The user then selects the number of buffers to use for each line to generate a key, which is fed into the AES encryption algorithm as its key. The result is an uncloneable key generated according to user specifications.

IV. Results

4.1. Analysis of the Configuration of the Buffers

Two configurations of the buffer selection are presented in the following section. The first is the linear selection of buffers, which is capable of placing between zero and seven buffers on the top or bottom input to the second NAND gate. The second configuration implements a cascading buffer design that has eight choices for the number of buffers placed on the path, utilizing between 0 and 63 buffers.

At least one buffer selection unit is always held to 0 while the other increases. This is done to keep the number of evaluations at a reasonable number, especially when testing designs that have more than eight choices of buffers.

Each output line is evaluated for every combination of buffers tested in this experiment. An output is considered "bad" if for all numbers of buffers, the same number of glitches is produced. The output line is unusable when all combinations produce the same glitch count because each line must have a choice of a '0' or a '1' to generate a bit of the key. There are five glitch units implemented on the Virtex-5 board, and each unit has 64 outputs. Thus, there are (5 x 64) = 320 total output lines tested in each design.

4.1.1. Linear Buffer Selection

Figure 15 shows the glitch circuit when configured with linear buffer selection. The buffer count is selected using the binary number representing the number of buffers desired. There are 15 total choices; 0 on top and bottom, 1 to 7 on the top, and 1 to 7 on the bottom. Testing reveals that 99 out of 320 total output lines, or 30.94%, are unusable. Each unit had between 16 and 27 “bad” outputs. A representative sampling of the results is shown in Table 3.

Table 3: Analysis of Linear Selection With Up to 7 Buffers

	Top Buffer Count:	0	1	2	3	4	5	6	7	0	0	0	0	0	0	0	Output Line
	Bottom Buffer Count:	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**Bad?**
Output Line #	**0**	0	100	0	0	0	100	0	0	0	100	100	0	0	100	98	
	1	100	100	100	100	100	0	100	100	0	100	100	100	100	100	100	
	2	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	bad
	3	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	bad
	4	0	0	0	100	0	0	0	100	0	0	100	0	0	0	100	
	5	0	0	0	100	0	0	0	100	0	0	0	0	0	0	100	
	6	0	0	100	100	0	0	100	100	100	0	0	0	0	0	0	
	7	100	0	100	100	100	0	100	100	100	0	0	100	100	0	0	
	8	0	100	0	0	0	100	0	0	0	0	0	0	0	0	0	
	9	0	0	0	100	0	0	0	100	0	0	100	0	0	0	100	
	10	0	0	0	0	0	0	0	0	0	0	100	0	0	0	100	
	11	100	100	100	100	100	100	100	0	100	100	100	100	100	100	100	
	12	0	4	0	0	0	0	0	0	0	0	0	0	0	0	0	bad

4.1.2. Cascading Buffer Selection

The circuit in Figure 16 illustrates the cascading buffer selection design. There are eight buffer selection choices on the top and bottom. They are 0, 1, 17, 22, 27, 47, 49, or 63. This is selectable using three SELECT bits. Table 2 explains the selection choices for this design. Each of the five units had between 31 and 38 unusable output lines. 172 or 53.75% of the outputs are “bad”. A portion of the analysis is displayed in Table 4 below.

Table 4: Analysis of Cascaded Selection With 8 Choices and Up to 63 Buffers

Top Buffer Count:		0	1	17	22	27	47	49	63	0	0	0	0	0	0	0	Output Line Bad?
Bottom Buffer Count:		0	0	0	0	0	0	0	0	1	17	22	27	47	49	63	
Output Line #	0	0	0	0	0	0	0	0	0	100	0	0	0	100	0	0	
	1	0	0	0	0	0	0	0	0	100	100	0	0	100	100	0	
	2	100	0	0	100	100	0	0	100	100	100	100	100	100	100	100	
	3	0	0	0	0	0	100	0	0	0	0	100	0	0	0	100	
	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	bad
	5	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	bad
	6	0	0	0	0	0	0	0	0	100	100	0	0	100	100	0	
	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	bad
	8	0	0	100	0	0	0	100	0	0	0	0	0	0	0	0	
	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	bad
	10	0	0	100	0	0	0	100	0	0	100	0	0	0	100	0	
	11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	bad
	12	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	bad

4.1.1. Summary of Results of Selection Configuration

The results of testing the buffer selection configuration demonstrate that the linear selection implementation produces less “bad” output lines. The linear selection method generates 14.79% more usable outputs. The next set of tests use this buffer selection configuration to find the optimum number of SELECT bits to implement in this design.

4.2. Analysis of the Number of SELECT bits

The previous section discusses the results of testing the buffer configuration. The linear selection design is clearly a better choice, as it outperforms its competitor by 14.79%. The following analysis determines the optimal number of buffers to insert into the logic circuit path by comparing the use of three, four, and six bits. As discussed in the buffer configuration analysis, the following analysis will consider an output line “bad” if all tested combinations of buffers generate the same number of glitches for that particular output.

4.2.1. Three SELECT bits

The results of tests using three SELECT bits in the linear buffer selection configuration are presented in Table 3. 30.94% of the output lines are considered usable.

4.2.2. Four SELECT bits

A sample of the results of using four SELECT bits in the linear buffer selection implementation is portrayed in Table 5. The table lists only the outcome of adding buffers into the bottom circuit path. Approximately 1 out of every 16 output lines is considered unusable. This constitutes 6.25% of all possible outputs being characterized as "bad".

Table 5: Analysis of Linear Selection With Up to 15 Buffers

Top Buffer Count:		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Output Line Bad?
Bottom Buffer Count:		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
Output Line #	0	0	0	0	0	100	0	100	0	0	0	0	0	100	0	100	
	1	100	5	100	100	100	100	100	90	100	2	100	100	100	100	100	
	2	100	0	100	0	100	0	100	0	100	0	100	100	100	100	100	
	3	0	100	100	0	100	0	100	100	0	100	100	0	100	0	100	
	4	100	100	100	0	100	100	100	0	100	0	100	100	100	100	100	
	5	0	0	100	0	100	100	100	0	100	100	100	0	100	80	100	
	6	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	
	7	0	100	0	0	100	0	100	100	0	100	0	0	100	0	100	
	8	0	0	0	100	0	0	0	0	0	0	0	0	0	0	0	
	9	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	
	10	0	0	0	100	100	100	100	100	100	100	100	100	100	100	100	
	11	100	97	100	0	100	0	100	100	100	100	100	0	100	0	100	
	12	0	0	0	0	0	0	0	0	0	0	100	0	0	0	100	
	13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	15	100	100	100	100	0	100	0	100	100	100	100	100	0	100	0	
	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	bad

4.2.3. Six SELECT bits

The results of testing the use of six SELECT bits reveal that 1.56% of all output lines are unusable. A representative selection of these results is depicted in Table 6.

Since there are 127 tested buffer counts, there are too many to show even a complete set of buffer counts, so a set of 12 consecutive counts are depicted in Table 6.

Table 6: Analysis of Linear Selection With Up to 63 Buffers

	Top Buffer Count:	0	0	0	0	0	0	0	0	0	0	0	0	# 100's	# 0's	IF at least 1 "0" and 1 "100"
	Bottom Buffer Count:	52	53	54	55	56	57	58	59	60	61	62	63			
Output Line #	0	100	100	100	100	100	100	100	100	100	100	100	100	105	21	T
	1	0	0	0	100	0	0	0	0	0	0	0	100	102	26	T
	2	100	0	100	0	100	0	100	0	100	0	0	100	111	16	T
	3	100	100	100	100	0	100	0	100	100	100	100	100	97	31	T
	4	100	100	100	100	100	100	100	100	100	100	100	100	104	24	T
	5	100	100	100	100	100	100	100	100	100	100	100	100	112	16	T
	6	100	100	100	100	100	100	100	100	100	100	100	100	108	20	T
	7	100	100	100	100	0	100	0	100	100	100	100	100	92	32	T
	8	100	100	100	100	100	100	100	100	100	100	100	100	97	30	T
	9	100	100	100	100	100	100	100	100	100	100	100	100	97	29	T
	10	100	100	100	100	0	0	0	0	100	100	100	100	82	46	T
	11	100	100	100	100	100	100	100	100	100	100	100	100	116	12	T
	12	100	100	0	100	100	100	100	100	100	100	100	100	85	41	T
	13	0	0	0	0	0	0	0	0	0	0	0	0	65	63	T
	14	0	0	0	0	0	0	0	0	0	0	0	0	60	64	T
	15	0	0	0	0	0	0	0	0	0	0	0	0	32	96	T
	16	0	0	0	0	0	0	0	0	0	0	0	0	72	56	T
	17	0	0	0	0	0	0	0	0	0	0	0	0	20	108	T
	18	0	0	0	0	0	0	0	0	0	0	0	0	21	107	T
	19	0	0	0	0	0	0	0	0	0	0	0	0	72	56	T
	20	100	0	100	0	100	0	100	0	100	0	0	0	91	36	T
	21	0	0	0	0	0	0	0	0	0	0	0	0	56	72	T
	22	0	0	0	0	0	0	0	0	0	0	0	0	0	128	FALSE

4.2.1. Summary of the Analysis of the Number of SELECT bits

The percent of unusable output lines is 30.94%, 6.25%, and 1.56% corresponding to three, four, and six SELECT bits respectively. The number of usable outputs increases by 24.69% when going from three to four bits, which implements twice as many buffers. When increasing the number of SELECT bits from four to six, only a gain of 4.69% in good outputs is seen by quadrupling the amount of logic gates.

The optimum implementation depends on how important space is, and also how many usable output lines are needed by the user. Each additional SELECT bit increases the number of logic gates by a power of two. The increase in logic needed by six bits as opposed to four bits is not an efficient use of space; however, the increase from three to four bits increases the total usable lines by almost 25%. The optimal choice for most users would likely be the four bit design. This would create approximately 300 output lines to use in generating a key, which would give 172 unused output lines after using 128 of them for a key. This gives the user more than double the amount of required bits for a key. It would also make it even more difficult for an adversary to be able to replicate the key since they would have to guess the correct bits and from which outputs those bits were taken.

4.3. Analysis of Key Generation

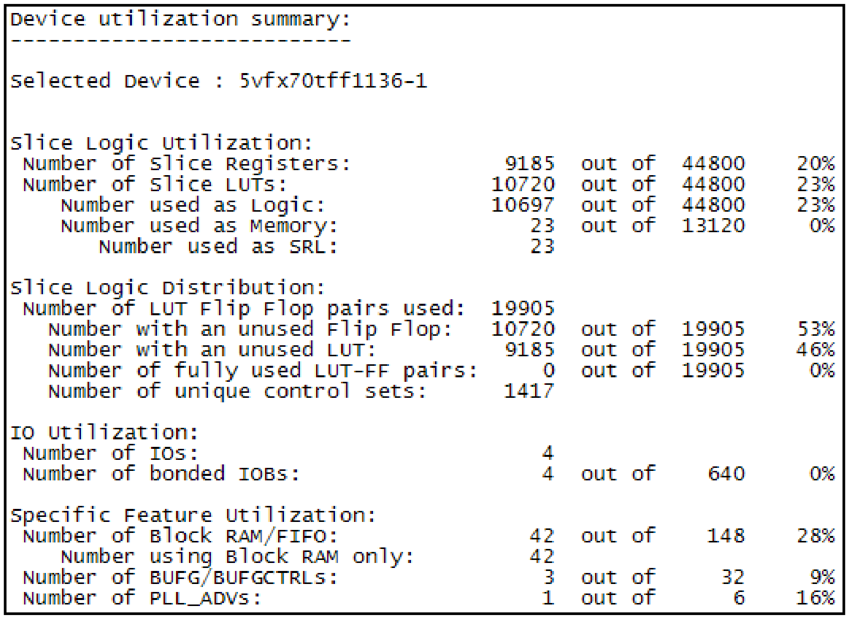
The analysis of the key generation includes testing the implementation, stability and the distinguishability of the glitch counts on each output line. The implementation tests the success or failure of implementing the AES algorithm. The key for AES is taken from the outputs of the glitch units. The stability of the glitches is a measure of how repeatable after the counts are each successive time the profiling algorithm is run. The glitch counts must be the same every time the code is executed in order to generate the same key. The distinguishability of the Uncloneable Key Generation System guarantees that the glitch counts are unique when the same code is run on another board.

4.3.1. Implementation of AES

Each of output line of a glitch unit corresponds to a simple glitch circuit. There are 64 glitch circuits comprising a glitch unit. The Uncloneable Key Generation System is composed of five glitch units. This creates a total of 320 output lines to specify a bit of the AES key. This overabundance of outputs is created to compensate for some output lines to be unusable and also makes it more difficult for an adversary to guess the key.

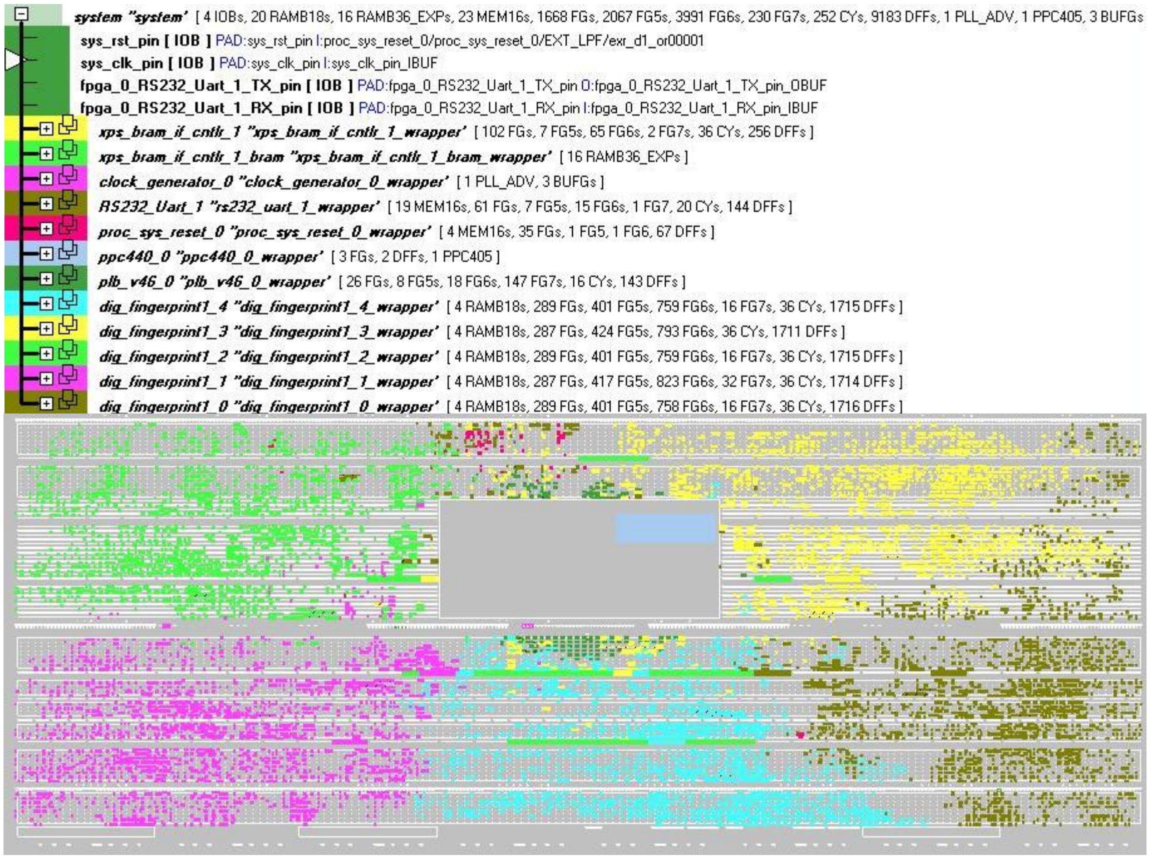
The result of the combination of these two entities for creating a key for AES encryption is a success. The Uncloneable Key Generation System operates at 125.00MHz. The user must first profile each output of the five glitch units and record the value of the glitch count on each of the 320 output lines at all buffer counts. This information is used to specify each bit of the key. To generate a bit of a particular key, the user must find an output line that is usable (has at least one ‘0’ and one ‘1’ glitch count at one of the buffer counts), then enter the glitch unit number, output line number, and the number of buffers that will produce the necessary bit value.

A picture showing the layout of five glitch units on a Virtex-5 board is shown in Figure 21, and the corresponding Device Utilization Summary is depicted in



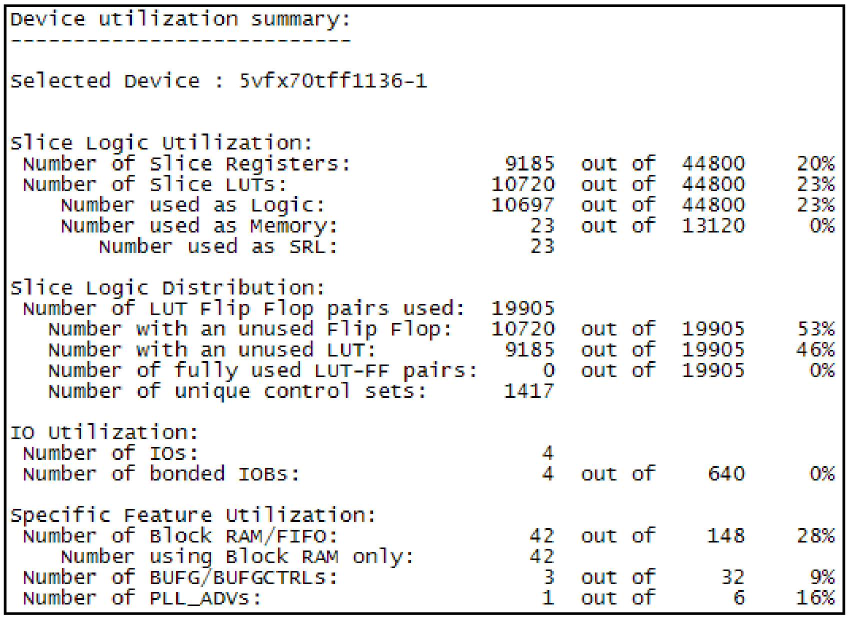
```
Device utilization summary:
---------------------------

Selected Device : 5vfx70tff1136-1


Slice Logic Utilization:
 Number of Slice Registers:              9185  out of  44800    20%
 Number of Slice LUTs:                  10720  out of  44800    23%
    Number used as Logic:               10697  out of  44800    23%
    Number used as Memory:                 23  out of  13120     0%
       Number used as SRL:                 23

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:    19905
   Number with an unused Flip Flop:     10720  out of  19905    53%
   Number with an unused LUT:            9185  out of  19905    46%
   Number of fully used LUT-FF pairs:       0  out of  19905     0%
   Number of unique control sets:        1417

IO Utilization:
 Number of IOs:                             4
 Number of bonded IOBs:                     4  out of    640     0%

Specific Feature Utilization:
 Number of Block RAM/FIFO:                 42  out of    148    28%
    Number using Block RAM only:           42
 Number of BUFG/BUFGCTRLs:                  3  out of     32     9%
 Number of PLL_ADVs:                        1  out of      6    16%
```

Figure 22. The layout of the Uncloneable Key Generation System is a combination of five glitch units and an AES module for encryption. The AES module uses approximately the same amount of resources as all five glitch units combined, which is clearly shown in Figure 23. As a result, the instantiation of the Uncloneable Key Generation System consumes slightly more than double the amount of resources as the five glitch units. Figure 24 displays the Device Utilization Summary for the Uncloneable Key Generation System.



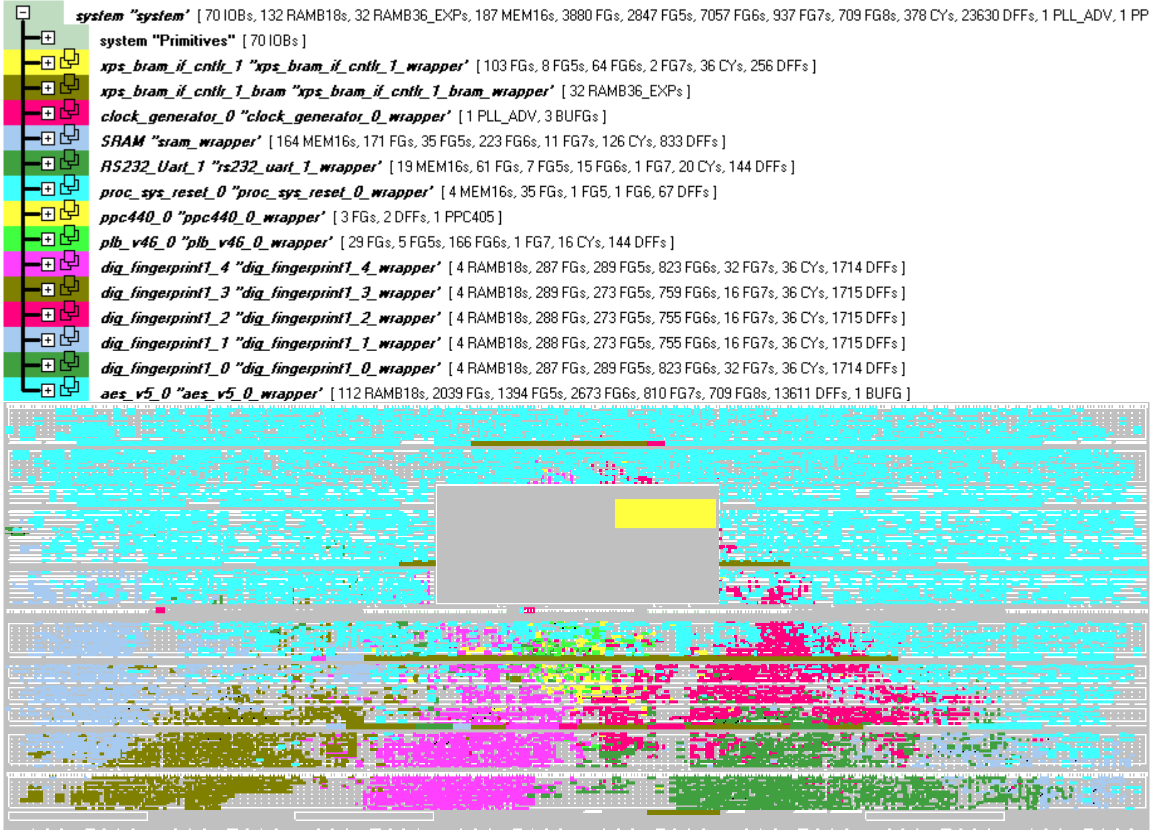
```
system "system' [ 4 IOBs, 20 RAMB18s, 16 RAMB36_EXPs, 23 MEM16s, 1668 FGs, 2067 FG5s, 3991 FG6s, 230 FG7s, 252 CYs, 9183 DFFs, 1 PLL_ADV, 1 PPC405, 3 BUFGs
  sys_rst_pin [ IOB ] PAD:sys_rst_pin I:proc_sys_reset_0/proc_sys_reset_0/EXT_LPF/exr_d1_or00001
  sys_clk_pin [ IOB ] PAD:sys_clk_pin I:sys_clk_pin_IBUF
  fpga_0_RS232_Uart_1_TX_pin [ IOB ] PAD:fpga_0_RS232_Uart_1_TX_pin O:fpga_0_RS232_Uart_1_TX_pin_OBUF
  fpga_0_RS232_Uart_1_RX_pin [ IOB ] PAD:fpga_0_RS232_Uart_1_RX_pin I:fpga_0_RS232_Uart_1_RX_pin_IBUF
    xps_bram_if_cntlr_1 "xps_bram_if_cntlr_1_wrapper' [ 102 FGs, 7 FG5s, 65 FG6s, 2 FG7s, 36 CYs, 256 DFFs ]
    xps_bram_if_cntlr_1_bram "xps_bram_if_cntlr_1_bram_wrapper' [ 16 RAMB36_EXPs ]
    clock_generator_0 "clock_generator_0_wrapper' [ 1 PLL_ADV, 3 BUFGs ]
    RS232_Uart_1 "rs232_uart_1_wrapper' [ 19 MEM16s, 61 FGs, 7 FG5s, 15 FG6s, 1 FG7, 20 CYs, 144 DFFs ]
    proc_sys_reset_0 "proc_sys_reset_0_wrapper' [ 4 MEM16s, 35 FGs, 1 FG5, 1 FG6, 67 DFFs ]
    ppc440_0 "ppc440_0_wrapper' [ 3 FGs, 2 DFFs, 1 PPC405 ]
    plb_v46_0 "plb_v46_0_wrapper' [ 26 FGs, 8 FG5s, 18 FG6s, 147 FG7s, 16 CYs, 143 DFFs ]
    dig_fingerprint1_4 "dig_fingerprint1_4_wrapper' [ 4 RAMB18s, 289 FGs, 401 FG5s, 759 FG6s, 16 FG7s, 36 CYs, 1715 DFFs ]
    dig_fingerprint1_3 "dig_fingerprint1_3_wrapper' [ 4 RAMB18s, 287 FGs, 424 FG5s, 793 FG6s, 36 CYs, 1711 DFFs ]
    dig_fingerprint1_2 "dig_fingerprint1_2_wrapper' [ 4 RAMB18s, 289 FGs, 401 FG5s, 759 FG6s, 16 FG7s, 36 CYs, 1715 DFFs ]
    dig_fingerprint1_1 "dig_fingerprint1_1_wrapper' [ 4 RAMB18s, 287 FGs, 417 FG5s, 823 FG6s, 32 FG7s, 36 CYs, 1714 DFFs ]
    dig_fingerprint1_0 "dig_fingerprint1_0_wrapper' [ 4 RAMB18s, 289 FGs, 401 FG5s, 758 FG6s, 16 FG7s, 36 CYs, 1716 DFFs ]
```

Figure 21: Layout of Five Glitch Units



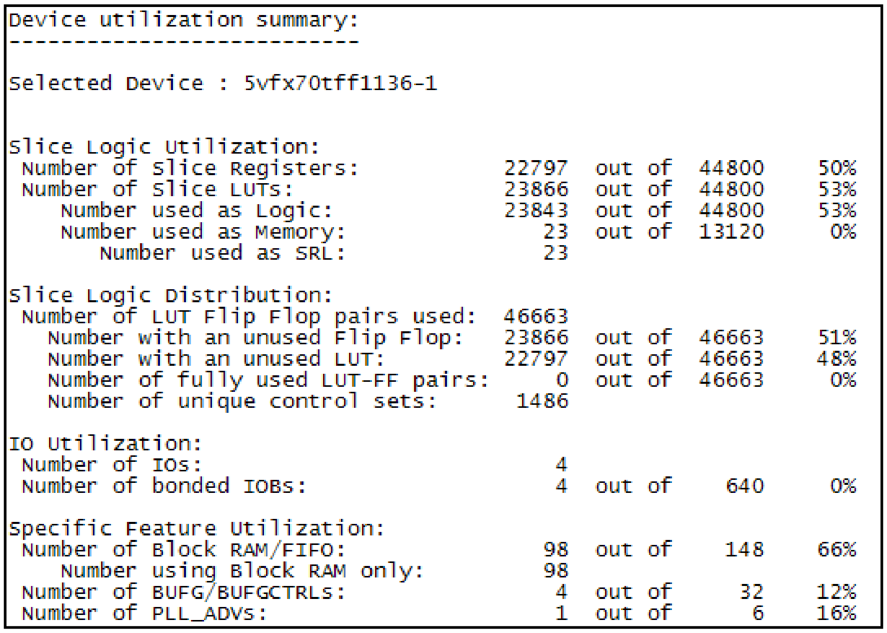
```
Device utilization summary:
---------------------------

Selected Device : 5vfx70tff1136-1


Slice Logic Utilization:
 Number of Slice Registers:            9185  out of  44800    20%
 Number of Slice LUTs:                10720  out of  44800    23%
    Number used as Logic:             10697  out of  44800    23%
    Number used as Memory:               23  out of  13120     0%
       Number used as SRL:               23

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:  19905
   Number with an unused Flip Flop:   10720  out of  19905    53%
   Number with an unused LUT:          9185  out of  19905    46%
   Number of fully used LUT-FF pairs:     0  out of  19905     0%
   Number of unique control sets:      1417

IO Utilization:
 Number of IOs:                           4
 Number of bonded IOBs:                   4  out of    640     0%

Specific Feature Utilization:
 Number of Block RAM/FIFO:               42  out of    148    28%
    Number using Block RAM only:         42
 Number of BUFG/BUFGCTRLs:                3  out of     32     9%
 Number of PLL_ADVs:                      1  out of      6    16%
```

Figure 22: Device Utilization Summary of Five Glitch Units



```
system ”system’ [ 70 IOBs, 132 RAMB18s, 32 RAMB36_EXPs, 187 MEM16s, 3880 FGs, 2847 FG5s, 7057 FG6s, 937 FG7s, 709 FG8s, 378 CYs, 23630 DFFs, 1 PLL_ADV, 1 PP
  system "Primitives" [ 70 IOBs ]
  xps_bram_if_cntlr_1 ”xps_bram_if_cntlr_1_wrapper’ [ 103 FGs, 8 FG5s, 64 FG6s, 2 FG7s, 36 CYs, 256 DFFs ]
  xps_bram_if_cntlr_1_bram ”xps_bram_if_cntlr_1_bram_wrapper’ [ 32 RAMB36_EXPs ]
  clock_generator_0 ”clock_generator_0_wrapper’ [ 1 PLL_ADV, 3 BUFGs ]
  SRAM ”sram_wrapper’ [ 164 MEM16s, 171 FGs, 35 FG5s, 223 FG6s, 11 FG7s, 126 CYs, 833 DFFs ]
  RS232_Uart_1 ”rs232_uart_1_wrapper’ [ 19 MEM16s, 61 FGs, 7 FG5s, 15 FG6s, 1 FG7, 20 CYs, 144 DFFs ]
  proc_sys_reset_0 ”proc_sys_reset_0_wrapper’ [ 4 MEM16s, 35 FGs, 1 FG5, 1 FG6, 67 DFFs ]
  ppc440_0 ”ppc440_0_wrapper’ [ 3 FGs, 2 DFFs, 1 PPC405 ]
  plb_v46_0 ”plb_v46_0_wrapper’ [ 29 FGs, 5 FG5s, 166 FG6s, 1 FG7, 16 CYs, 144 DFFs ]
  dig_fingerprint1_4 ”dig_fingerprint1_4_wrapper’ [ 4 RAMB18s, 287 FGs, 289 FG5s, 823 FG6s, 32 FG7s, 36 CYs, 1714 DFFs ]
  dig_fingerprint1_3 ”dig_fingerprint1_3_wrapper’ [ 4 RAMB18s, 289 FGs, 273 FG5s, 759 FG6s, 16 FG7s, 36 CYs, 1715 DFFs ]
  dig_fingerprint1_2 ”dig_fingerprint1_2_wrapper’ [ 4 RAMB18s, 288 FGs, 273 FG5s, 755 FG6s, 16 FG7s, 36 CYs, 1715 DFFs ]
  dig_fingerprint1_1 ”dig_fingerprint1_1_wrapper’ [ 4 RAMB18s, 288 FGs, 273 FG5s, 755 FG6s, 16 FG7s, 36 CYs, 1715 DFFs ]
  dig_fingerprint1_0 ”dig_fingerprint1_0_wrapper’ [ 4 RAMB18s, 287 FGs, 289 FG5s, 823 FG6s, 32 FG7s, 36 CYs, 1714 DFFs ]
  aes_v5_0 ”aes_v5_0_wrapper’ [ 112 RAMB18s, 2039 FGs, 1394 FG5s, 2673 FG6s, 810 FG7s, 709 FG8s, 13611 DFFs, 1 BUFG ]
```

Figure 23: Layout of Uncloneable Key Generation System



```
Device utilization summary:
---------------------------

Selected Device : 5vfx70tff1136-1


Slice Logic Utilization:
 Number of Slice Registers:            22797  out of  44800    50%
 Number of Slice LUTs:                 23866  out of  44800    53%
    Number used as Logic:              23843  out of  44800    53%
    Number used as Memory:                23  out of  13120     0%
       Number used as SRL:                23

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:   46663
   Number with an unused Flip Flop:    23866  out of  46663    51%
   Number with an unused LUT:          22797  out of  46663    48%
   Number of fully used LUT-FF pairs:      0  out of  46663     0%
   Number of unique control sets:       1486

IO Utilization:
 Number of IOs:                            4
 Number of bonded IOBs:                    4  out of    640     0%

Specific Feature Utilization:
 Number of Block RAM/FIFO:                98  out of    148    66%
    Number using Block RAM only:          98
 Number of BUFG/BUFGCTRLs:                 4  out of     32    12%
 Number of PLL_ADVs:                       1  out of      6    16%
```

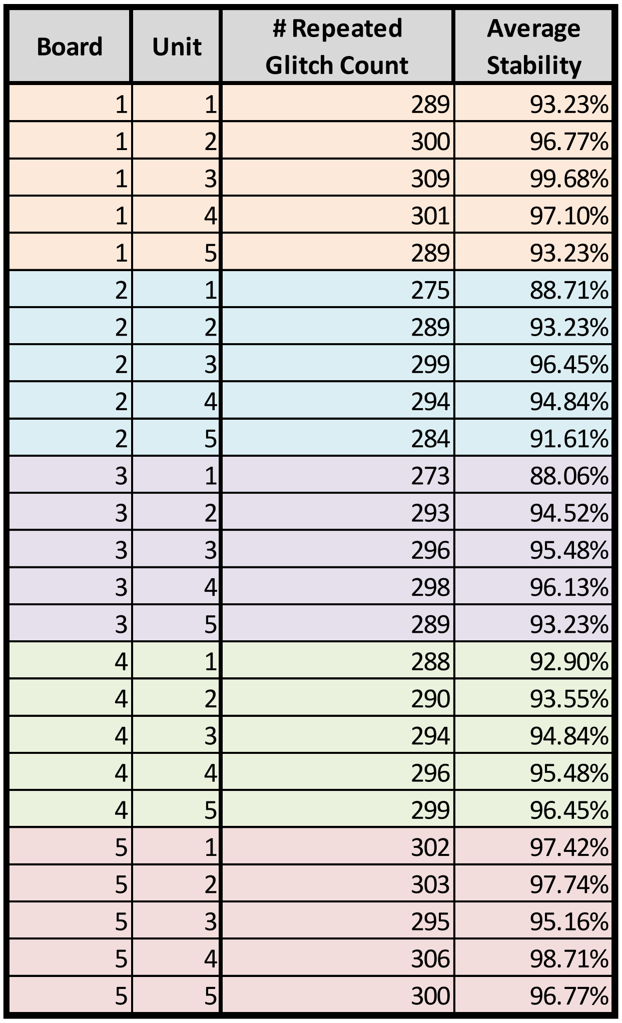
Figure 24: Device Utilization Summary of Uncloneable Key Generation System

4.3.2. Stability

Stability tests confirm that the glitch counts for each output line for a Virtex-5 FPGA are the same each time the C code is executed. Preliminary testing showed that temperature is a pertinent factor in returning stable glitch counts. The time between print statements and memory access executions change the heat generated within the circuit. The code that generates and records glitch counts must be nearly identical to ensure that the glitch count will stay the same when the circuits are profiled and during the generation of the key.

After profiling each of the five units instantiated on five different Virtex-5 boards, results show that stability of each unit varies from 88.06% to 99.68%. The outcome of the stability tests for each unit is displayed in Table 7. The average stability over all 25 units is 94.85% with a standard deviation of 2.77%.

Table 7: Percent Stability for Each Glitch Unit



Board	Unit	# Repeated Glitch Count	Average Stability
1	1	289	93.23%
1	2	300	96.77%
1	3	309	99.68%
1	4	301	97.10%
1	5	289	93.23%
2	1	275	88.71%
2	2	289	93.23%
2	3	299	96.45%
2	4	294	94.84%
2	5	284	91.61%
3	1	273	88.06%
3	2	293	94.52%
3	3	296	95.48%
3	4	298	96.13%
3	5	289	93.23%
4	1	288	92.90%
4	2	290	93.55%
4	3	294	94.84%
4	4	296	95.48%
4	5	299	96.45%
5	1	302	97.42%
5	2	303	97.74%
5	3	295	95.16%
5	4	306	98.71%
5	5	300	96.77%

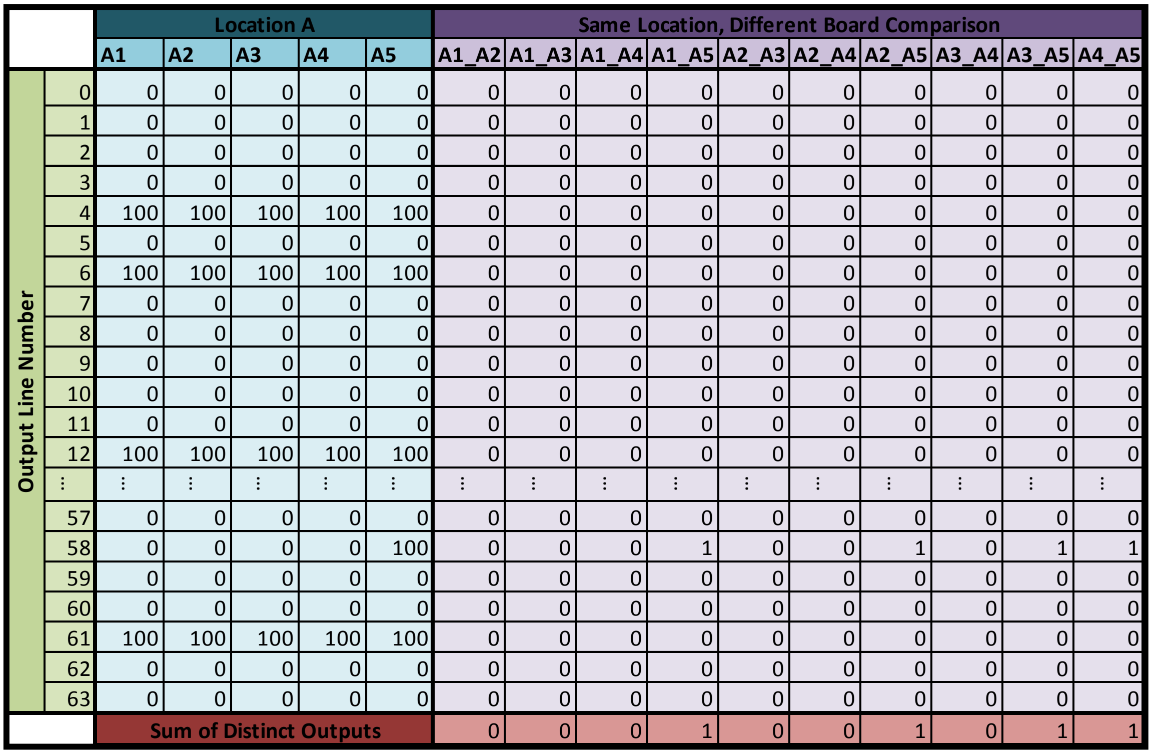
4.3.3. Distinguishability

The distinguishability tests on this method are accomplished in order to measure the degree to which one unit differs from another one generated on a different FPGA. Each glitch unit has 64 output lines that generate a binary bit. There are five glitch units (A, B, C, D, and E) instantiated on each of the five Virtex-5 FPGAs. The placement and routing of each unit is identical, therefore, the distinguishability between each location A, B, C, D, and E will be evaluated against its matching unit on the other boards at each buffer count. Essentially, the distinguishability of the location of each output line for each number of buffers is evaluated. There are 25 total units under test. Each unit is given a letter and number identifier. For example, the glitch unit A on board 1 is called A1, and unit A on board 2 uses the identifier A2, and so on. Units A1, A2, A3, A4, and A5 are compared against each other for each output line of each buffer quantity. The same is true for units B, C, D, and E. For each output line of a unit with the same letter identifier, there are ten comparisons. Each of the ten comparisons is summed over the 64 output locations to determine the Hamming distance between each pair of units at the same location. This determines the number of distinct output lines at the same location at a specific buffer count.

The number of distinct output lines between each pair of boards is summed for each buffer count. A sample of the glitch count of all location A comparisons is shown in Table 8 for location A on each of the five boards. The figure does not show all 64 output lines.

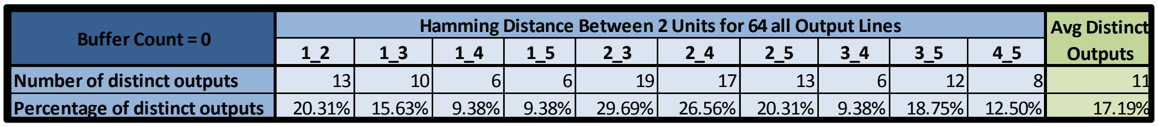
The average distinguishability over each of the ten pair-wise comparisons at each buffer quantity is recorded. A sample of the sum of the ten pair-wise comparisons at buffer quantity=0 is displayed in Table 9. There are 31 different buffer counts, consisting of 0 buffers on both the top and bottom circuit paths, 1 to 15 buffers on top while holding the bottom at 0, and 1 to 15 on the bottom while holding the top at 0. These 31 averages, one for each buffer count, are averaged together to determine the overall distinguishability of the Uncloneable Key Generation System.

Table 8: Distinguishability Sample for Location A Comparisons



		Location A					Same Location, Different Board Comparison									
		A1	A2	A3	A4	A5	A1_A2	A1_A3	A1_A4	A1_A5	A2_A3	A2_A4	A2_A5	A3_A4	A3_A5	A4_A5
Output Line Number	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	4	100	100	100	100	100	0	0	0	0	0	0	0	0	0	0
	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	6	100	100	100	100	100	0	0	0	0	0	0	0	0	0	0
	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	12	100	100	100	100	100	0	0	0	0	0	0	0	0	0	0
	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	57	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	58	0	0	0	0	100	0	0	0	1	0	0	1	0	1	1
	59	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	60	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	61	100	100	100	100	100	0	0	0	0	0	0	0	0	0	0
	62	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	63	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		Sum of Distinct Outputs					0	0	0	1	0	0	1	0	1	1

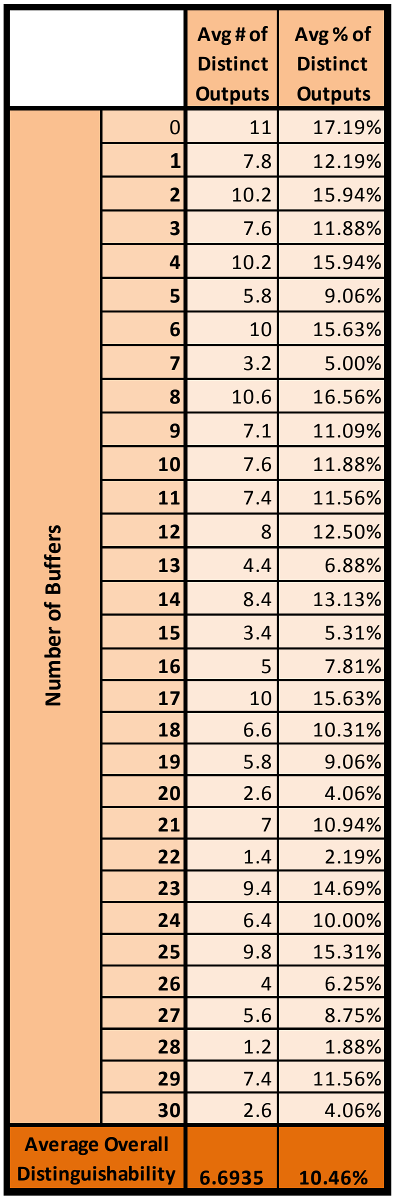
Table 9: Sample of Average Number of Distinct Outputs at Buffer Count = 0



Buffer Count = 0	Hamming Distance Between 2 Units for 64 all Output Lines										Avg Distinct Outputs
	1_2	1_3	1_4	1_5	2_3	2_4	2_5	3_4	3_5	4_5	
Number of distinct outputs	13	10	6	6	19	17	13	6	12	8	11
Percentage of distinct outputs	20.31%	15.63%	9.38%	9.38%	29.69%	26.56%	20.31%	9.38%	18.75%	12.50%	17.19%

Table 10 displays a summary of the results of distinguishability tests. The table shows the average number and percentage of distinct outputs out of the five matching units A, B, C, D, and E for a given buffer count. The overall distinguishability is 10.46%. The standard deviation is 4.41%.

Table 10: Distinguishability Results



		Avg # of Distinct Outputs	Avg % of Distinct Outputs
Number of Buffers	0	11	17.19%
	1	7.8	12.19%
	2	10.2	15.94%
	3	7.6	11.88%
	4	10.2	15.94%
	5	5.8	9.06%
	6	10	15.63%
	7	3.2	5.00%
	8	10.6	16.56%
	9	7.1	11.09%
	10	7.6	11.88%
	11	7.4	11.56%
	12	8	12.50%
	13	4.4	6.88%
	14	8.4	13.13%
	15	3.4	5.31%
	16	5	7.81%
	17	10	15.63%
	18	6.6	10.31%
	19	5.8	9.06%
	20	2.6	4.06%
	21	7	10.94%
	22	1.4	2.19%
	23	9.4	14.69%
	24	6.4	10.00%
	25	9.8	15.31%
	26	4	6.25%
	27	5.6	8.75%
	28	1.2	1.88%
	29	7.4	11.56%
	30	2.6	4.06%
Average Overall Distinguishability		6.6935	10.46%

4.4. Results Summary

This chapter describes the results of testing the configuration of the selectable buffers, the number of SELECT bits used on the selectable buffer unit, and the analysis of the stability and distinguishability of this method's key generation capabilities.

Two configurations for the buffer selection module were tested. The linear buffer selection design inserts a quantity of buffers equal to the binary equivalent of the number on the SELECT line. The cascaded design inserts one of eight different quantities of buffers (0, 1, 17, 22, 27, 47, 49, or 63) into the glitch circuit path. Tests show that the linear configuration is capable of producing 14.79% more usable output lines than the cascaded design.

The linear buffer selection configuration is used to test the use of three, four, and six bits on the SELECT input line. Results demonstrate that as the number of SELECT bits increase, so do the number of usable outputs. Four bits give nearly a 25% improvement over three bits, while six bits give less than a 5% gain over the use of four bits.

The implementation of AES using the modified digital fingerprint method is fully operational. The user can specify each bit of the 128-bit key by entering a glitch unit number, the output line number, and the number of buffers to insert into the glitch circuit path.

The key generation analysis details the results of testing the stability and distinguishability of generating a secure, uncloneable key. Stability tests confirm that the glitch count for each of the 25 units can be repeated with an average probability of 94.85%. The tests on the distinguishability of the output lines show that the glitch counts are distinct 10.46% of the time when comparing the five units to its matching unit on the other FPGAs at the same buffer count.

V. Conclusions and Recommendations

In the previous chapter, the results of experimenting on the buffer configuration and number of buffer SELECT bits were presented, as well as the stability and distinguishability test results of key generation using the modified Digital Fingerprinting Method with AES encryption. This chapter details the conclusions reached due to these results.

5.1. Buffer Configuration Conclusion

In this research study, two methods of choosing the number of buffers inserted into a logic path are investigated. These designs are evaluated based solely on the number of usable output lines that are produced when using three buffer SELECT bits, which equates to eight different choices for the total amount of buffers on a wire. The linear buffer selection configuration created 22.81% more usable outputs than the cascaded circuit.

The optimal configuration of buffers uses linear buffer selection with four bits, which select between 0 and 15 buffers. Four bits generate close to a 25% gain in usable outputs as compared to using three bits. Employing six bits rather than four bits provides less than a 5% increase in usable output lines.

5.2. Key Generation Conclusion

The stability, or repeatability, of the glitch counts taken on the same board are very stable, with close to 95% of the counts being repeatable over 10 runs. The distinguishability of the glitch counts is slightly over 10%.

5.3. Recommendations for Future Research

5.3.1. Force routing to be more efficient in Xilinx

The routing delays between design components cause the output glitch counts to be unpredictable. Future work in this area could investigate a way to force Xilinx to use an efficient method of wiring adjacent modules together. One option would be to find a way to manually place and route individual components.

5.3.2. AES Decryption

This research effort implements only the encryption portion of the AES algorithm. It is only a proof of concept that using the basic ideas behind the Digital Fingerprinting method can be used to generate a secure uncloneable key. Implementing the decryption part of AES will provide a complete encryption system to send and receive sensitive information.

5.3.3. Make Uncloneable Key Generation System More Efficient

There are several modifications that can be made to the existing Uncloneable Key Generation System to make it more efficient. In an actual implementation of this system, the space used by the design should be minimized as much as possible. Currently, the number of usable output lines far exceeds the amount needed to produce a key. One overall method of decreasing space is to reduce either the number of SELECT bits or the number of glitch units instantiated in VHDL.

Another way to utilize less hardware space is to reduce the width of the input shift registers. Only two bit combinations are used to generate glitches, while each input shift register is 64 bits long. It would save a significant amount of space to reduce the three input shift registers on all 64 input lines of the five glitch circuits from 64 to 2.

5.4. Conclusions Summary

The optimal configuration of buffers uses linear selection with four SELECT bits, which insert between 0 and 15 buffers on a delay path. Test results show that key generation reliably provides a stable and distinguishable key for AES encryption. The Uncloneable Key Generation System can also be used as a new way to generate a DF.

Bibliography

[1] Xilinx, “FPGA vs. ASIC,” 2010. [Online]. Available: http://www.xilinx.com/company/gettingstarted/fpgavasic.htm#pcs. [Accessed March 3, 2010].

[2] O. -. Standaert, E. Peeters, G. Rouvroy and J. -. Quisquater. (2006, An overview of power analysis attacks against field programmable gate arrays. *Proceedings of the IEEE 94; 94(2),* pp. 383-394.

[3] H. Patel. (2009). A top down approach to creating a digital fingerprint to uniquely identify field programmable gate arrays. M. Sc. Thesis. Air Force Institute of Technology: USA.

[4] G. E. Suh and S. Devadas. (2007, Physical unclonable functions for device authentication and secret key generation. *Design Automation Conference, 2007. DAC '07. 44th ACM/IEEE* pp. 9-14.

[5] S. S. Kumar, J. Guajardo, R. Maes, G. -. Schrijen and P. Tuyls. (2008, Extended abstract: The butterfly PUF protecting IP on every FPGA. *Hardware-Oriented Security and Trust, 2008. HOST 2008. IEEE International Workshop on* pp. 67-70.

[6] A. Agarwal, D. Blaauw and V. Zolotov. (2003, Statistical timing analysis for intra-die process variations with spatial correlations. *Computer Aided Design, 2003. ICCAD-2003. International Conference on* pp. 900-907.

[7] G. S. May, S. M. Sze, *Fundamentals of Semiconductor Fabrication*, Hoboken, NJ: Wiley, 2004, pp. 124–134.

[8] Y. Taur, T. H. Ning. *Fundamentals of Modern VLSI Devices,* New York: Cambridge University Press, 1998, pp. 257-271.

[9] H. Mahmoodi and K. Roy, “Estimation of Delay Variations due to Random-Dopant Fluctuations in Nanoscale CMOS Circuits,” IEEE Journal of Solid-State Circuits, vol. 40, no. 9, Sep., pp. 1787-1796, 2005.

[10] M. Majzoobi, F. Koushanfar, and M. Potkonjak. (2008, Lightweight secure PUFs. *Computer-Aided Design, 2008. ICCAD 2008. IEEE/ACM International Conference on* pp.670-673.

[11] Dan Jiang and Cheun Ngen Chong. (2008, Anti-counterfeiting using phosphor PUF. *Anti-Counterfeiting, Security and Identification, 2008. ASID 2008. 2nd International Conference on* pp. 59-62.

[12] E. Ozturk, G. Hammouri and B. Sunar. (2008, Physical uncloneable function with tristate buffers. *Circuits and Systems, 2008. ISCAS 2008. IEEE International Symposium on* pp. 3194-3197.

[13] D. Roy, J.H. Klootwijk, N. Verheagh, H. Roosen and R. Wolters. (2009, Comb capacitor structures for on-chip physical uncloneable function. *Semiconductor Manufacturing, IEEE Transactions on*, vol. 22, no. 1, pp. 99-102.

[14] J. W. Crouch. (2008). Digital fingerprinting of field programmable gate arrays. M. Sc. Thesis. Air Force Institute of Technology: USA.

[15] W. Trappe, L. C. Washington, *Introduction to Cryptography with Coding Theory*, Upper Saddle River, NJ: Pearson Prentice Hall, 2006, pp. 113–160.

[16] B.E. Stine, D.S. Boning, J.E. Chung, D.J. Ciplickas, J.K. Kibarian. (1998, Simulating the impact of pattern-dependent poly-CD variation on circuit performance," *Semiconductor Manufacturing, IEEE Transactions on* , vol.11, no.4, pp. 552-556, Nov 1998
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=728551&isnumber=15704

[17] B.F. Romanescu, M. E. Bauer, S. Ozev, and D. J. Sorin. 2007. VariaSim: simulating circuits and systems in the presence of process variability. *SIGARCH Comput. Archit. News* 35, 5 (Dec. 2007), 45-48. URL: http://doi.acm.org/10.1145/1360464.1360465

[18] J. Salzmann, F. Sill, D. Timmermann, "Algorithm for Fast Statistical Timing Analysis," *System-on-Chip, 2007 International Symposium on*, pp. 1-4, 20-21 Nov. 2007
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4427424&isnumber=4427418

[19] A.A. Gaffar, J. A. Clarke G. A. Constantinides, "Modeling of glitch effects in FPGA based arithmetic circuits," Field Programmable Technology, 2006. *FPT 2006. IEEE International Conference on*, pp. 349-352, Dec. 2006
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4042467&isnumber=4042097

[20] USA. National Security Administration. *CNSS Policy No. 15, Fact Sheet No. 1: National Policy on the Use of the Advanced Encryption Standard (AES) to Protect National Security Systems and National Security Information*. Committee on National Security Systems, 2003.

[21] USA. National Institute of Standards and Technology. *Federal Information Processing Standards Publication 197 Announcing the ADVANCED ENCRYPTION STANDARD (AES).* National Institute of Standards and Technology, 2001.

[22] Xilinx, “KEEP_HIERARCHY Architecture Support,” 2010. [Online]. Available: http://www.xilinx.com/itp/xilinx7/books/data/cgd/cgd0110_71.html. [Accessed February 25, 2010].

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. REPORT DATE *(DD-MM-YYYY)* 25-03-2010	**2. REPORT TYPE** Master's Thesis	**3. DATES COVERED** *(From – To)* September 2008 – March 2010

TITLE AND SUBTITLE

Utilizing the Digital Fingerprint Method for Secure Key Generation

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Anilao, Jennifer C., 2d Lt, USAF

5d. PROJECT NUMBER
JON 08-248

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S)
Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/ENG)
2950 Hobson Way, Building 640
WPAFB OH 45433-8865

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/GE/ENG/10-02

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Tony Kim Air Force Research Laboratory Anti-Tamper Software Protection
2241 Avionics Circle
WPAFB OH 45433-7301
(Ph: 937-320-9068 ext 124, E-mail: tony.kim@wpafb.af.mil)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This research examines a new way to generate an uncloneable secure key by taking advantage of the delay characteristics of individual transistors. The user profiles the circuit to deduce the glitch count of each output line for each number of selectable buffers added to the circuit. The user can then use this information to generate a specific glitch count on each output line, which is passed to an encryption algorithm as its key.

The results detail tests of two configurations for adding a selectable amount of buffers into each glitch circuit in order to induce additional delay. One configuration adds up to seven buffers that is equivalent to the binary digits used on the three SELECT lines of a multiplexer. The second, referred to as the cascaded design, has eight different quantities of selectable buffers, but they all connect to one multiplexer. Each successive line connects to the previous line and adds a certain number of buffers. The linear selection implementation produces almost 15% more usable output lines over the cascaded design, where a usable line is defined as one that has at least one '1' and one '0' glitch count in response to every buffer count.

Tests were also performed to determine the optimal number of buffers added to each output using the linear buffer selection configuration. Using three input bits to the buffer unit produced 30.94% usable outputs. Four bits generated nearly 25% more usable outputs, while the use of six bits gave less than a 5% improvement over four bits.

The average repeatability of the glitch count is 94.85% using this method. The overall distinguishability of the generated glitch counts for each output line is 10.46%.

15. SUBJECT TERMS
FPGA, digital fingerprint, circuit ID, key generation, stability, distinguishability

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Dr. Yong Kim
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			**19b. TELEPHONE NUMBER** *(Include area code)*
U	U	U	UU	77	(937) 255-3636, ext 4620 (yong.kim@afit.edu)

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39-18